

Data Science e Tecnologie per le Basi di Dati

Sommario

1.	<i>Intro</i>	2
	Le 5 V dei big data	2
	Processo di data science	3
	Knowledge Discovery Process	3
	Problemi della data science	5
2.	<i>Data warehouse</i>	5
	Data Warehouse	5
3.	<i>Progettazione di un data warehouse</i>	7
	Business Dimensional Lifecycle	8
	Progettazione Concettuale	9
	Progettazione Logica	12
4.	<i>Analisi dei dati</i>	14
	Strumenti di interfaccia	14
	Analisi OLAP	14
5.	<i>Viste materializzate</i>	19
	Progettazione fisica	19
	Alimentazione del DW	20
6.	<i>Data lakes</i>	24
7.	25
8.	<i>Data preprocessing</i>	25
	Dati	25
	Tecniche di data preprocessing	26
	Similarità e dissimilarità	29
9.	<i>Regole di associazione</i>	30
	Estrazione delle regole di associazione	30
	Generazione degli itemset frequenti	31
10.	<i>Classificazione</i>	35
	Albero decisionale	35
	Creazione del training set e test set	45
	Validazione dei modelli	46
11.	<i>Clustering</i>	46
	Algoritmi di clustering	47
	Validazione degli algoritmi di clustering	52
12.	<i>Introduzione ai DBMS</i>	54
	Transazioni	54
13.	<i>Buffer Manager</i>	55
14.	<i>Accesso fisico ai dati</i>	57
15.	<i>Progettazione fisica</i>	60
	Selezione delle strutture dati	61
16.	<i>Ottimizzazione delle query</i>	64

Ottimizzazione algebrica	65
Query tree	66
Individuazione del piano di esecuzione.....	68
L'ottimizzatore di Oracle	69
Hint in Oracle	70
17. <i>Gestione della concorrenza</i>	71
Locking gerarchico.....	75
Deadlock.....	76
18. <i>Reliability manager</i>.....	77
Gestione del ripristino.....	79
19. <i>DBMS distribuiti</i>.....	80
Architetture client/server	81
Sistemi di database distribuiti.....	81
Design di database distribuiti.....	82
Tecnologie per i DBMS distribuiti.....	83
X-Open-DTP	85
Beyond relational databases (NoSQL).....	85
MongoDB	89
20. <i>ElasticSearch</i>.....	92

1. Intro

I **big data** sono dati la cui **dimensione** (grandi volumi di dati che scalano nel tempo), **diversità** (eterogenei e di diverso tipo) e **complessità** (sia temporale che spaziale) richiedono nuove architetture, tecniche, algoritmi e analitica che li gestiscano e che possano estrarre valore e conoscenza da essi (mediante viste personalizzate in base al target).

Esempio: Gestione delle emergenze

In questo caso vengono integrati dati di natura diversa (sensori, immagini, video, analisi degli stream social, ...) eterogenei sia dal punto di vista spaziale che da quello temporale. L'aggregazione risulta complicata e viene svolta mediante tecniche di **data fusion**.

I dati memorizzati e storiciizzati all'interno di un **data center** devono essere elaborati prima di essere mostrati ad utenti con necessità di visualizzazione diversa.

Le 5 V dei big data

Le **V** dei big data sono:

- **Volume:** La dimensione dei dati memorizzati cresce esponenzialmente nel tempo
- **Velocità:** È necessario che l'elaborazione dei dati sia veloce per assicurare tempestività
- **Varietà:** I dati possono essere di formato, tipo e struttura diverse
- **Veridicità:** I dati raccolti devono essere sicuri e affidabili
- **Valore:** I dati raccolti devono servire a creare valore e un vantaggio commerciale

Processo di data science

Il processo di **data science** è composto di quattro fasi:



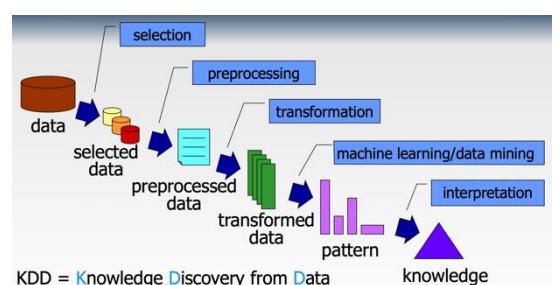
- **Generazione**: i dati possono essere generati in diversi modi:
 - **Registrazione passiva**: dati strutturati come transazioni, vendite...
 - **Generazione attiva**: dati non strutturati o semi strutturati come testi, immagini, video...
 - **Produzione automatica**: dati dipendenti dal contesto come quelli generati da sensori IoT
- **Acquisizione**
 - **Raccolta**: pull-based (web crawler), push-based (video sorveglianza e click stream)
 - **Trasmissione**: trasferimento verso un data center mediante collegamenti veloci
 - **Preelaborazione**: integrazione, pulizia, eliminazione delle ridondanze
- **Memorizzazione**
 - **Infrastruttura di memorizzazione**: memorie locali (HDD, SSD) o di rete (DAS, NAS, SAN)
 - **Gestione dei dati**: file systems, struttura dati chiave-valore, DB column-oriented, DB a documenti (MongoDB)
 - **Modelli di programmazione**: Map Reduce, elaborazione degli stream ed elaborazione dei grafi
- **Analisi**
 - **Obiettivi**:
 - **Descrizione analitica**: caratterizzazione del dataset
 - **Analisi predittiva**: predizione sulla base di un modello predittivo generato
 - **Analisi prescrittiva**: ottimizzazione sulla base del modello predittivo
 - **Metodi**: analisi statistica, machine learning e data mining, text mining, pattern mining, analisi associativa, classificazione e regressione, clustering (modellazione fatta sulla base dei dati stessi, divisione in sottogruppi)

Knowledge Discovery Process

È possibile utilizzare **machine learning** e tecniche di **data science** per estrarre dai dati informazioni **implicite**, nascoste, sconosciute e potenzialmente **utili**. L'estrazione è automatica ed eseguita da algoritmi appropriati e le informazioni estratte sono rappresentate mediante modelli astratti chiamati **pattern**.

Il processo di estrazione di informazioni dai dati è detto **Knowledge Discovery Process**, esso è un processo iterativo.

Inizialmente viene analizzato un campione rappresentativo dei dati e vengono fatti test con diversi algoritmi e diverse configurazioni, quando si ottengono dei buoni risultati, l'elaborazione viene estesa a tutto il dataset.



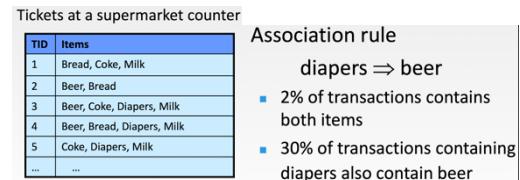
Preprocessing

La fase di **preprocessing** si divide in due micro-fasi:

- **Data cleaning**
 - Riduzione dell'effetto del rumore
 - Rimozione degli outliers
 - Risoluzione delle inconsistenze
- **Data integration**
 - Riconciliazione dei dati estratti da diverse sorgenti
 - Integrazione dei metadati
 - Individuazione e risoluzione dei conflitti fra diversi valori dei dati
 - Gestione della ridondanza

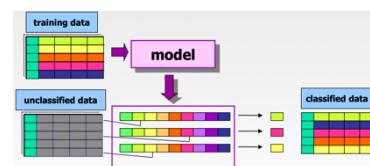
Tecniche di Data Science:

- **Regole di associazione:** è una tecnica utilizzabile su dati di qualsiasi natura e consiste nell'estrazione di correlazioni frequenti o pattern da una database transazionale.
È di tipo **supervised** in quanto abbiamo già una conoscenza pregressa dei dati.

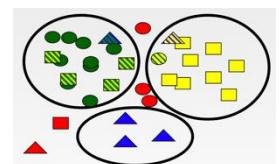


Una regola di associazione è composta da due parti:

- **Supporto:** percentuale di record che contiene entrambi gli oggetti
- **Confidenza:** Indica quanto è forte l'implicazione. Tra i record che contengono l'informazione a sinistra dell'associazione la percentuale di quelli che contengono anche quella a destra.
- **Classificazione:** è di tipo **supervised** in quanto prevede di avere una lista di dati già classificati che viene utilizzata per ottenere un modello che permette di categorizzare dati "nuovi" non classificati
- **Clustering:** è di tipo **unsupervised** e serve a determinare gruppi di dati simili (le label dei gruppi non sono note a priori) identificando eccezioni e outliers.



Può essere utilizzato per le definizione di un campione rappresentativo, dividendo prima i dati in sottogruppi e prendendo una percentuale di campioni da ogni gruppo.



- **Sequence mining:** vengono presi in considerazione i criteri di ordinamento dei dati
- **Serie temporali e dati geospatiali:** vengono considerate le informazioni relative a spazio e tempo (sensori)
- **Regression:** predizione di un valore continuo (quotazioni in borsa)
- **Rilevazione degli outliers**

Diversi tipi di data scientist

Per una corretta analisi dei dati sono richieste diverse **personalità**:

- **Data expert:** strutturazione e elaborazione dei dati
- **Data analyst:** data mining, statistiche e machine learning
- **Visualization expert:** visualizzazione grafica dei dati e story telling
- **Domain expert:** aiuta la comprensione del dominio dell'applicazione

- **Business expert:** compie le scelte di business mediante modelli di business e la comprensione dei dati analizzati

Problemi della data science

Un problema ancora irrisolto riguardo la data science è la presenza di bias nelle scelte prese dai meccanismi di machine learning, uno dei motivi di queste discriminazioni è il fatto che gli algoritmi di intelligenza artificiale si comportano come “black box” e per questo motivo le scelte che vengono fatte non sono accuratamente giustificate.

Per compensare queste mancanze si stanno sviluppano degli algoritmi di AI nell’ambito dell'**explainable AI**. Questi algoritmi sono basati su alberi decisionali, intrinsecamente più “chiari” e hanno diversi aspetti che li rendono più “chiari” rispetto a quelli tradizionali:

- **Utilizzo di alberi decisionali:** intrinsecamente più leggibili
- **Spiegazione del modello:** comprensione globale di come funziona il modello
- **Spiegazione delle singole predizioni:** spiegazione del perché è stata fatta una specifica predizione
- **Selezione di features interpretabili:** introduzione di criteri basati sull’interpretabilità nel design del modello

Esistono due tipi di **attributi** correlati al **bias** in un algoritmo di machine learning:

- **Attributo Bias:** un attributo direttamente discriminatorio (sesso, nazionalità)
- **Attributo Proxy:** è un attributo indirettamente discriminatorio (CAP, da cui si risale alla nazionalità di residenza)

2. Data warehouse

I **dati** di tipo **operativo** memorizzati da un’azienda possono costituire una miniera di informazioni utili al **supporto delle decisioni aziendali**.

L’utilizzo di **strumenti** per questo tipo di supporto permettono di analizzare lo stato dell’azienda e consente di prendere decisioni rapide e veloci.

La **Business Intelligence** è una disciplina atta al supporto alla **decisione strategica aziendale** e ha l’obiettivo di trasformare i dati aziendali in informazioni fruibili, per fare ciò si rende necessaria un’adeguata infrastruttura hardware e software che permetta di servire utenti di tipologia eterogenea.

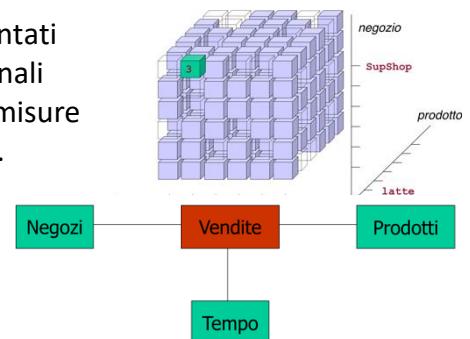
Data Warehouse

Un **Data Warehouse** è una base di dati per il supporto alle decisioni mantenuta separatamente dalle basi di dati operative dell’azienda, questo viene fatto sia per motivi di **prestazioni** (ricerche complesse ridurrebbero le performance delle transazioni operative) e motivi di **gestione dei dati** (qualità dei dati, dati integrati, consistenti, orientati ai soggetti di interesse, dipendenti dal tempo).

Rappresentazione dei DW

È possibile rappresentare un datawarehouse in due modi:

- **Rappresentazione multidimensionale:** i dati vengono rappresentati come un ipercubo con tre o più dimensioni in cui gli assi direzionali sono le entità coinvolte e ogni intersezione contiene una o più misure (quantità, importo, ...) relative a quelle determinate coordinate.
- **Rappresentazione relazionale (a stella):** le misure presenti nella tabella dei fatti sono collegate alle entità (dimensioni) attraverso delle relazioni.



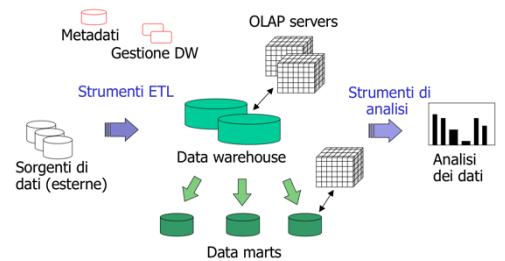
Analisi dei dati

È possibile analizzare i dati secondo diverse tecniche:

- **Analisi OLAP:** calcolo di funzioni aggregate complesse
- **Analisi dei dati mediante tecniche di data mining**
- **Presentazione:** rappresentazione strutturata dei risultati di una ricerca mediante diversi tipi di strumenti per la rappresentazione
- **Ricerca di motivazioni:** esplorazione dei dati mediante approfondimenti (drill down: aumento della precisione anni>mesi>settimane>giorni)

Elementi costruttivi di un data warehouse

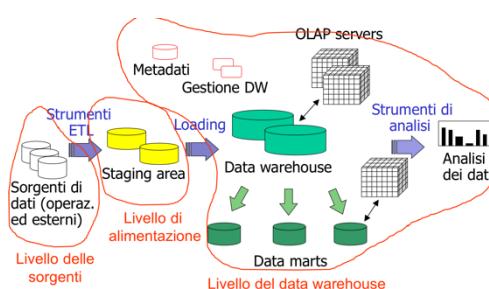
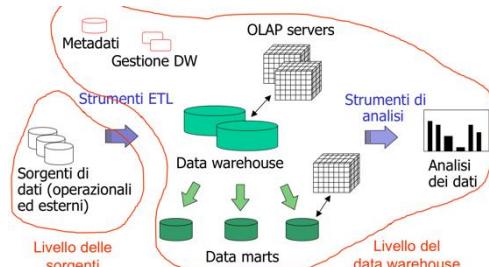
- **Sorgenti di dati:** sorgenti eterogenee esterne al warehouse (SI aziendali, tweet, excel)
- **Warehouse aziendale:** contiene sia la modellazione del dato, sia strutture OLAP che servono ad interagire con il dato
- **Data mart:** sottoinsieme dipartimentale focalizzato su un settore prefissato, può essere alimentato dal DW (copia fisica di una parte del DW) o alimentato direttamente dalle sorgenti (è in sé una parte di DW). Richiede un'attenta progettazione in modo da evitare futuri problemi di integrazione
- **Server OLAP:** può essere di diversi tipi:
 - **Server ROLAP (Relational OLAP):** DBMS relazionale esteso con aggregazioni complesse, non ha il vincolo sulla normalizzazione e può presentare ridondanze
 - **Server MOLAP (Multidimensional OLAP):** rappresenta i dati in forma matriciale, è utile per dati molto densi
 - **Server HOLAP (Hybrid OLAP):** memorizzazione relazionale/multidimensionale e lettura multidimensionale/relazionale
- **Strumenti ETL:** ETL è il processo di preparazione dei dati da introdurre nel DW e viene eseguito sia al primo popolamento che all'aggiornamento periodico.
Il processo di ETL è composto da 4 fasi:
 - **Estrazione:** acquisizione dei dati dalle sorgenti
 - **Pulitura:** operazioni volte al miglioramento della qualità dei dati (consistenza e correttezza)
 - **Trasformazione:** conversione dei dati dal formato operazionale a quello del DW (integrazione)



- **Caricamento:** propagazione degli aggiornamenti al data warehouse
- **Metadati:** sono informazioni relative al dato stesso (sorgenti, attributi e relazioni tra i dati). Possono essere di 3 tipi:
 - **Trasformazione e caricamento:** descrivono i dati sorgente e le trasformazioni necessarie
 - **Gestione dei dati:** descrivono la struttura dei dati presenti nel DW (dati derivati come le liste materializzate)
 - **Gestione delle query:** dati sulla struttura delle query e monitoraggio della loro esecuzione (codice SQL della query, uso di memoria e CPU, ...)

L'architettura di un DW può essere di due tipi:

- **A due livelli:** (sorgenti, DW) ha diversi vantaggi:
 - Disaccoppiamento dalle sorgenti: possibilità di gestire dati esterni al sistema OLTP
 - Facilità di gestione delle differenti granularità temporali dei dati operazionali e analitici
 - Separazione del carico transazionale da quello analitico
 - Necessita di svolgere la preparazione dei dati “on the fly”, se l'operazione non va a buon fine bisogna ricominciare dall'inizio, quindi i tempi devono essere abbastanza contenuti
- **A tre livelli:** (sorgenti, alimentazione, DW) viene introdotta una **staging area** (denominata anche Operational Data Store (ODS)) che si comporta da area di transito e consente la separazione dell'elaborazione ET dal caricamento L permettendo operazioni complesse di trasformazione e pulizia dei dati.
D'altro canto però questa architettura introduce **ulteriore ridondanza**.



3. Progettazione di un data warehouse

Durante la progettazione di un DW bisogna tener conto di diversi fattori di rischio:

- Aspettative elevate degli utenti
- Qualità dei dati e dei processi OLTP di partenza: i dati a disposizione possono essere incompleti o inaffidabili perché derivanti da processi aziendali non integrati e ottimizzati
- Gestione “politica” del progetto: accettazione del sistema da parte degli utenti finali e collaborazione con i “detentori” dell'informazione

Approcci di progettazione

La progettazione di un DW può avvenire secondo due modalità differenti:

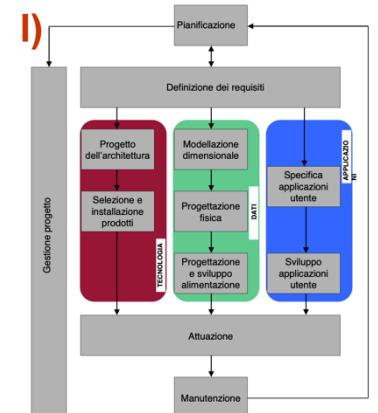
- **Approccio top-down:** il DW viene costruito nel suo insieme fin dall'inizio, questo approccio comporta una fase di progettazione complessa e dei costi e dei tempi significativi, non conviene in caso di realtà molto grandi
- **Approccio bottom-up:** il DW viene realizzato in maniera incrementale partendo dalla costruzione di data mart su settori aziendali strategici per poi estendere man mano il

sistema in tutte le aree dell'azienda. In questo caso la fase di progettazione è più semplice e i costi e i tempi della consegna dei primi data mart sono esigui

Business Dimensional Lifecycle

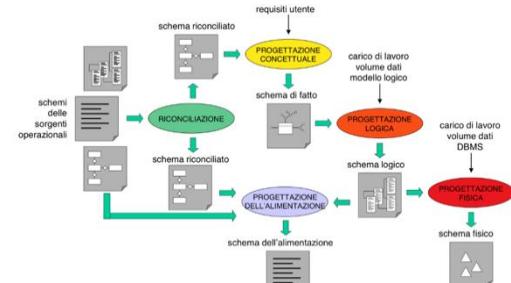
Il **Business Dimensional Lifecycle** rappresenta il flow di progettazione di un DW:

- **Definizione dei requisiti:** in questa fase vengono apprese due tipologie di requisiti:
 - **Requisiti utente:** appresi mediante intervista con gli attori di business
 - **Requisiti relativi ai dati e alle sorgenti**
- **Progettazione:**
 - **Tecnologia:** è necessario progettare l'architettura del sistema (hw) e vanno selezionati quali prodotti software utilizzare (sw)
 - **Dati:**
 - **Progettazione concettuale:** modellazione dimensionale mediante la quale devono essere definite le dimensioni di analisi, le metriche e le gerarchie delle dimensioni di analisi
 - **Progettazione fisica:** descrizione di come devono essere sviluppare le tabelle
 - Progettazione e sviluppo dei processi di ETL (caricamento iniziale e periodico)
 - **Applicazioni:** specifica e sviluppo di applicazioni per la generazione di report, l'esplorazione interattiva dei dati e elaborazione dei dati mediante tecniche di data science
- **Attuazione:** il sistema viene deployato, durante l'utilizzo potrebbero venir fuori nuovi requisiti che non sono stati considerati per i quali si torna in fase di progettazione



La progettazione di un data mart segue le seguenti fasi:

- **Caratterizzazione delle sorgenti:** le sorgenti possono presentarsi come una descrizione testuale, un modello logico o un modello ER
- **Riconciliazione:** consiste in una visione globale delle fonti
- **Progettazione concettuale:** creazione del modello di DW sulla base dei requisiti utente, il modello multidimensionale che ne viene fuori contiene le dimensioni di analisi, le misure e le diverse gerarchie sulle dimensioni
- **Progettazione logica:** elaborazione di uno schema logico denormalizzato (che contiene le tabelle e le relazioni fra di esse) che tenga conto del volume di dati relativo alle query che verranno eseguite sul DW per prevedere un sistema ottimizzato
- **Progettazione del processo di ETL:** produce uno schema di alimentazione maggiormente composto da script
- **Progettazione fisica:** definizione di strutture fisiche accessorie che possano ridurre il tempo di esecuzione delle operazioni (indici per ridurre i tempi di lettura e viste materializzate per ridurre i tempi di elaborazione di group by e join)



Analisi dei requisiti

L'**analisi dei requisiti** consiste nell'effettuare delle interviste mirate agli utilizzatori del sistema (business users) e a coloro che detengono le informazioni delle sorgenti (amministratori del sistema informativo) e nello scegliere uno o più data mart che faranno parte del progetto pilota. I data mart scelti devono essere strategici prevedendo un ritorno economico per l'azienda e devono essere alimentati da poche sorgenti affidabili.

Distinguiamo due tipi di requisiti:

- **Requisiti applicativi:** (fatti, dimensioni e granularità temporale) descrivono gli eventi di interesse, l'intervallo di storicizzazione e il carico di lavoro che caratterizzano i dati relativi al settore dell'azienda in cui vogliamo implementare il data mart, analizzando le dimensioni di interesse
- **Requisiti strutturali:**
 - **Periodicità dell'alimentazione:** ogni quanto tempo i dati vengono trasferiti dalle sorgenti al DW
 - **Spazio disponibile:** stima dello spazio richiesto dal DW considerando anche la sparsità dei dati e dello spazio previsto per garantire maggiore efficienza, offerta dalla progettazione fisica (strutture accessorie come indici e viste materializzate)
 - **Tipo di architettura del sistema:** la scelta del numero di livelli è legata alla complessità del processo di ETL e al volume di dati da trasferire periodicamente
 - **Pianificazione del deployment:** fase di avviamento del progetto seguita da una fase di formazione degli utilizzatori del sistema

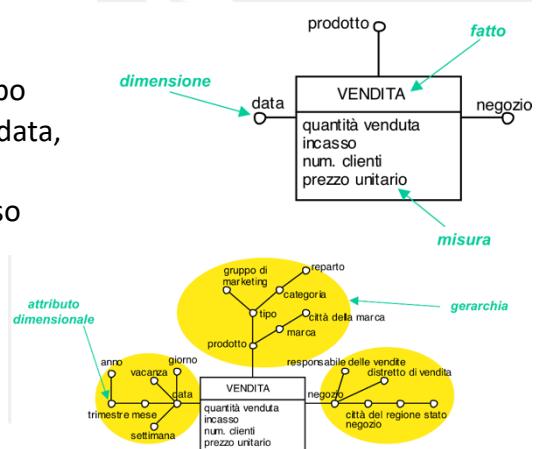
Progettazione Concettuale

La **progettazione concettuale** è la rappresentazione di alto livello di come i dati vengono organizzati, viene superato il modello ER perché troppo **rigido** e fortemente legato alla **normalizzazione**.

Utilizziamo il **Dimesional Fact Model** che definisce schemi di fatto che modellano i fatti, le dimensioni, le misure e le gerarchie e rappresenta un modello grafico che offre una documentazione utile per la revisione dei requisiti e per comprendere a posteriori le motivazioni delle scelte fatte.

- **Fatto:** è un elemento che modella un insieme di eventi di interesse (vendite, spedizioni, reclami) in relazione al tempo
- **Dimensione:** descrive un coordinata di analisi di un fatto (data, negozio, prodotto) e rappresenta una relazione n:n
- **Misura:** descrive una proprietà numerica di un fatto, spesso oggetto di operazioni di aggregazione (incasso)
- **Gerarchia:** rappresenta una relazione di generalizzazione tra un sottoinsieme di attributi di una dimensione e rappresenta una relazione 1:n.

Ogni pallino bianco rappresenta un **attributo dimensionale**, più esso è vicino al fatto e più è ritenuto specifico.



Gli attributi temporali vengono indicati come:

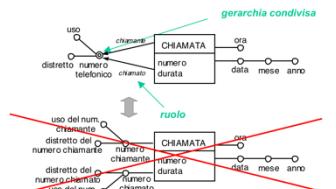
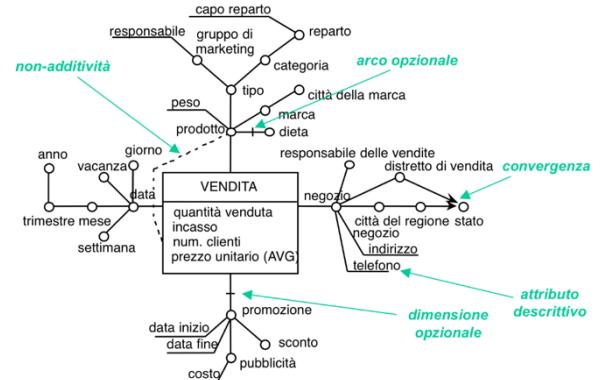
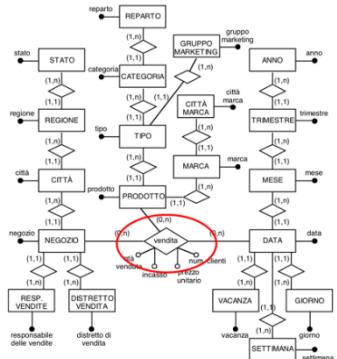
- **Anno:** 2022
- **Mese:** ottobre 2022 contiene sempre l'indicazione dell'anno
- **Trimestre:** [1-4] 2022 anch'esso contiene l'indicazione sull'anno e un'indicazione cardinale relativa al trimestre
- **Settimana:** [1-52] 2022 con indicazione relativo al numero della settimana
- **Giorno:** 5 ottobre 2022

Osservando l'ER corrispondente notiamo come ogni gerarchia viene rappresentata come un'entità ed una relazione.

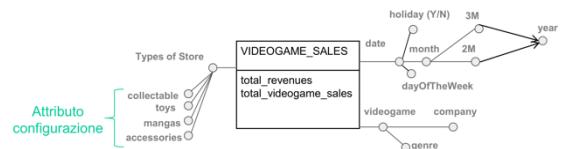
Costrutti avanzati

Utilizziamo diversi **costrutti avanzati**:

- **Non-additività:** rappresentata da un arco tratteggiato identifica una metrica e una dimensione non aggregabili, guardando l'esempio non è possibile stabilire il numero di clienti relativi a più prodotti perché in uno stesso scontrino ci possono essere prodotti diversi. Se prod_1 ha 100 clienti (100 vendite) e prod_2 ne ha 200, la somma totale dei clienti dovrebbe essere 300 ma in realtà ci saranno delle sovrapposizioni dovute al fatto che su uno scontrino ci possono essere sia prod_1 che prod_2.
- **Arco opzionale:** rappresenta un attributo non disponibile per tutti i record, sostituisce l'indicazione della cardinalità nel modello ER
- **Convergenza:** due rami che convergono verso lo stesso attributo
- **Attributo descrittivo:** (Non sono delle dimensioni di analisi) sono attributi che non vengono interessati da operazioni come la group by ma rappresentano del continuo informativo dell'attributo a cui essi sono collegati
- **Dimensione opzionale:** rappresenta una dimensione non disponibile per tutti i record del fatto
- **Gerarchia condivisa:** usare una gerarchia per due dimensioni di analisi (freccia)
- **Arco multiplo:** rappresenta una molti a molti fra due attributi dimensionali e può essere rappresentato in due modi:
 - L'arco multiplo si trova fra la dimensione e il fatto: la tabella dei fatti presenterà n record per lo stesso fatto dovuti al numero delle configurazioni della dimensione. Se il numero di attributi multipli è limitato è preferibile usare un'attributo configurazione.
 - La dimensione diventa un attributo di una nuova dimensione che rappresenta un **gruppo** di configurazioni della dimensione iniziale, viene utilizzato per snellire la tabella dei fatti



- **Attributo configurazionale:** rappresenta un attributo multivalore che è solitamente caratterizzato da pochi valori distinti ($<=10$), è rappresentabile mediante enumerazione dei valori possibili, rappresentati in tabella come colonne che possono assumere un valore booleano, una seconda alternativa è quella di considerare l'attributo configurazione come un normale attributo (che nello schema logico diventa un attributo della tabella padre della gerarchia), oppure una chiave esterna alla tabella che contiene tutte le configurazioni possibili (dipende dalla numerosità delle configurazioni).



Aggregazione

L'**aggregazione** è il processo di calcolo del valore di misure a granularità meno fine di quella presente nello schema di fatto originale, la riduzione del livello di dettaglio viene ottenuta risalendo la gerarchia mentre il valore finale viene ottenuto mediante operatori di aggregazione standard (SUM, MIN, MAX, AVG, COUNT)

Classificazione delle **misure in base all'aggregabilità**:

- **Additive**
- **Non additive:** non aggregabili lungo una gerarchia mediante l'operatore di somma
- **Non aggregabili**

Classificazione delle **misure in base al tipo**:

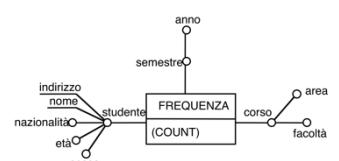
- **Misure di flusso:** possono essere valutate cumulativamente alla fine di un periodo di tempo e sono aggregabili mediante tutti gli operatori standard (quantità di prodotti venduti, numero di clienti)
- **Misure di livello:** sono valutate in specifici istanti di tempo e non sono additive lungo la dimensione del tempo (livello di inventario, saldo del conto corrente)
- **Misure unitarie:** sono valutate in specifici istanti di tempo non additive lungo nessuna dimensione ed espresse in termini relativi (prezzo unitario di un prodotto)

Gli **operatori di aggregazione** possono essere di tre **tipi**:

- **Distributivi:** è sempre possibile il calcolo di aggregati da dati di livello di dettaglio maggiore (SUM, MIN, MAX)
- **Algebrici:** il calcolo degli aggregati da dati di livello di dettaglio maggiore è possibile in presenza di misure aggiuntive di supporto (AVG: richiede l'informazione sul count)
- **Olistici:** non è possibile il calcolo di aggregati da dati di livello di dettaglio maggiore (moda, mediana)

Schema di fatto vuoto

Uno schema di fatto **vuoto** è uno schema di fatto in cui il fatto rappresenta il verificarsi di un evento e non contiene misure al suo interno, è utile per conteggiare gli eventi accaduti e per rappresentare gli eventi non accaduti (insieme di copertura che è l'insieme di tutti i possibili casi, nell'esempio della frequenza alle lezioni l'insieme di copertura è il calendario delle lezioni). La misura può essere solo una count



Variazione dei dati nel tempo

La variazione delle dimensioni o dei dati relativi ad un record, nel tempo, può essere gestita in tre modalità diverse:

- **Sovrascrittura (Rappresentazione del tempo di tipo 1):** il nuovo dato sovrascrive il valore attuale “proiettando” la situazione attuale nel passato, non rimane traccia del cambiamento avvenuto e del valore precedente.
- **Nuova istanza incorrelata (Rappresentazione del tempo di tipo 2):** per ogni variazione della dimensione viene creata una nuova istanza nella dimensione correlando i nuovi eventi alla nuova istanza (gli eventi sono partizionati in base alla variazioni degli attributi dimensionali).
- **Nuova istanza collegata alle vecchie (Rappresentazione del tempo di tipo 3):** rispetto al caso precedente vengono introdotti:
 - Una **coppia di timestamp** (inizio e fine) per la gestione della validità del dato
 - Un **attributo** che consenta l’identificazione della **sequenza** di variazione di una specifica istanza.

Carico di lavoro

Durante la progettazione è fondamentale definire il **carico di lavoro** che il sistema deve poter gestire.

Il possibile carico reale previsto viene calcolato mediante **reportistica standard** e **stime** discusse con gli **utenti**, queste stime devono prevedere anche il **successo** del sistema (che porta ad un’aumento del numero di utenti nel tempo) e la variazione della tipologia delle **interrogazioni** nel tempo. Dopo l’avviamento del sistema esso deve essere monitorato e migliorato (fase di **tuning**).

Volume dei dati

In questa fase è importante anche la **stima dello spazio** che il data mart necessita per la memorizzazione dei **dati** (mediante il DFM) e delle **strutture accessorie**. La previsione viene fatta mediante l’aggregazione di vari fattori:

- Numero di eventi per ogni fatto
- Numero di valori distinti negli attributi delle gerarchie
- Lunghezza degli attributi
- Intervallo di memorizzazione dei dati
- Sparsità: il numero degli eventi accaduti non corrisponde a tutte le possibili combinazioni delle dimensioni

Progettazione Logica

La **progettazione logica** consiste nella rappresentazione dello schema delle tabelle mediante il **modello relazionale** ROLAP (i modelli MOLAP e HOLAP non sono molto utilizzati).

Lo schema delle tabelle viene fatto tenendo a mente la progettazione concettuale, il carico di lavoro, il volume dei dati e i vincoli di sistema (direttive per ETL).

La progettazione logica, in questo caso consente la **ridondanza** dei dati e la **denormalizzazione** delle tabelle.

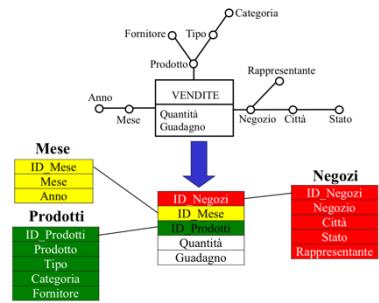
Schema a stella

Lo **schema a stella** viene realizzato così:

- **Dimensioni:** viene creata una tabella per ogni dimensione, con una chiave primaria generata artificialmente (surrogata), che contiene tutti gli attributi senza esplicitare le gerarchie fra di loro

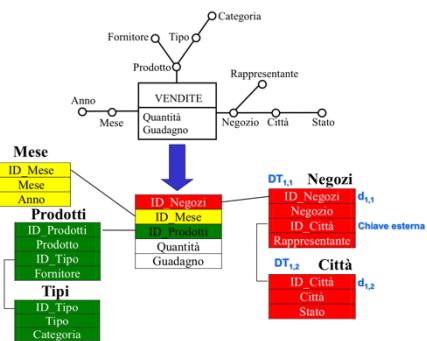
- **Fatti:** viene creata una tabella per ogni schema di fatto, con una chiave primaria costituita dalla combinazione delle chiavi esterne delle dimensioni e le misure sono rappresentate come attributi della tabella.

Una dimensione **opzionale**, in base alle specifiche di progetto può essere inclusa o meno nella chiave primaria (nel secondo caso è possibile avere solo un valore possibile per l'attributo opzionale relativamente a quella chiave primaria)



Schema snowflake (sconsigliato)

Lo schema a **snowflake** invece separa alcune dipendenze funzionali frazionando i dati di una dimensione in più tabelle, lo spazio necessario viene ridotto ma la diminuzione della ridondanza rallenta la fase di ricostruzione dell'informazione della dimensione per colpa delle join. Questo schema è solitamente **sconsigliato** in quanto la riduzione di spazio occupato è scarsamente benefica, **si usa più frequentemente quando una parte della gerarchia è condivisa da più dimensioni**.



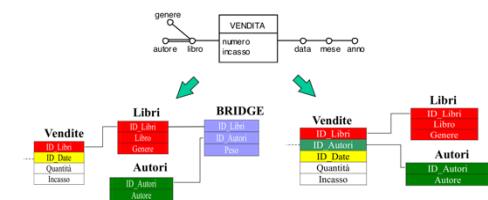
Archi multipli

Le query che caratterizzano un arco **multiplo** possono essere di tipo:

- **Pesato:** considerano il peso dell'arco multiplo (il peso è un attributo della bridge table)
- **Di impatto:** il valore aggregato viene calcolato senza considerare il peso

Gli archi multipli possono essere realizzati mediante due soluzioni:

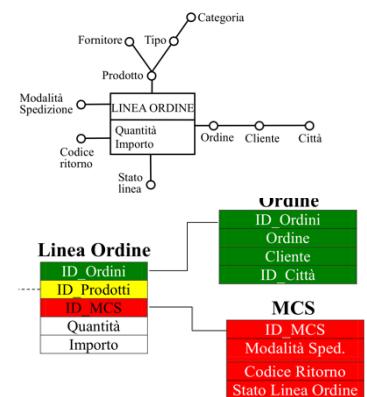
- **Bridge table:** viene aggiunta una tabella che modella la relazione molti a molti contenente un nuovo attributo che consente di pesare la partecipazione delle tuple nella relazione. È adatto per query pesate ma non per query di impatto.
Il calcolo del peso deve essere svolto in fase di ETL e possibili modifiche successive risultano più complicate.
- **Push down:** (sconsigliato perché separa il concetto del fatto su più righe. In alcuni casi potrebbe essere errato applicarlo in base alla traccia) l'arco multiplo viene integrato nella tabella dei fatti così da aggiungere una nuova dimensione.
C'è aumento di ridondanza (minore costo di esecuzione) nella tabella dei fatti e la gestione delle metriche relative al fatto andrebbero opportunamente separate.
 - È difficile eseguire interrogazioni di impatto
 - Il calcolo del peso viene fatto durante l'alimentazione (il peso deve essere distribuito nei record)
 - Le modifiche successive sono difficoltose in quanto è difficile risalire al peso iniziale di un gruppo di righe



Dimensione degenera

Una dimensione **degenera** è una dimensione costituita da un solo attributo ed è rappresentabile mediante 3 soluzioni distinte:

- Viene trattata come una dimensione **normale**, cioè consente l'aggiunta futura di gerarchie
- Viene **integrata** nella tabella dei fatti (per attributi di dimensione contenuta – es pochi kB) e nella rispettiva chiave primaria
- **Junk dimension**: in presenza di più dimensioni degeneri esse vengono raggruppate figurando come unica dimensione. La nuova dimensione deve poter rappresentare tutte le possibili combinazioni delle dimensioni al suo interno, questa soluzione viene quindi attuata solo per cardinalità limitate del dominio degli attributi.



4. Analisi dei dati

L'**analisi dei dati** viene svolta principalmente mediante tre strumenti:

- Calcolo di funzioni **aggregate** lungo una o più dimensioni
- Operazioni di **confronto**
- Analisi mediante tecniche di **data mining**

Strumenti di interfaccia

Il DW può essere **interrogato** mediante strumenti di vario tipo:

- Ambiente controllato di query
- Strumenti specifici di query e generazione rapporti
- Strumenti di data mining (esplorazione del dato e creazione di modelli predittivi più accurati)

Ambiente controllato di query

Un **ambiente controllato di query** consente di effettuare **ricerche, analisi e rapporti** con struttura **prefissata**, i risultati vengono rappresentati mediante grafici e report **personalizzati**.

In questo caso è possibile introdurre **elementi specifici** del settore economico considerato ed è necessario lo sviluppo di **codice ad hoc**.

Ambiente di query ad hoc

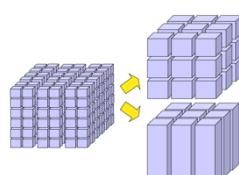
In un **ambiente di query ad hoc** è possibile definire interrogazioni OLAP arbitrarie e anche complesse, progettate al momento dall'utente; è utile quando i report predefiniti non sono abbastanza.

Una sessione di lavoro OLAP permette raffinamenti successivi alla stessa query.

Analisi OLAP

L'**analisi OLAP** permette l'analisi dei dati mediante diverse operazioni che possono essere combinate tra loro nella stessa query e eseguite in sequenza in una stessa sessione OLAP:

- **Roll up**: riduzione del dettaglio dei dati:
 - Aumentando il livello di gerarchia (cambio delle clausole della group by)
 - Eliminando una delle dimensioni presenti (dalla group by)



Metrics	Dollar Sales	Customer Region	North-East	Mid-Atlantic	South-East	Central	South	North-West	South-West	England	France	Germany	Canada
Category													
Electronics	\$ 1,200	Jan	\$ 751	\$ 30	\$ 660	\$ 2,040	\$ 1,110	\$ 440	\$ 1,000	\$ 303	\$ 210		
Food	\$ 250	Feb	\$ 250	\$ 800	\$ 975	\$ 160	\$ 582	\$ 744	\$ 310	\$ 799	\$ 118	\$ 357	
Furniture	\$ 640	Mar	\$ 244	\$ 148	\$ 250	\$ 1,085	\$ 2,961	\$ 650	\$ 1,240	\$ 119	\$ 142	\$ 96	
Gifts	\$ 100	Apr	\$ 97	\$ 198	\$ 160	\$ 1,080	\$ 568	\$ 470	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	
Health & Beauty	\$ 1,350	May	\$ 245	\$ 936	\$ 159	\$ 664	\$ 526	\$ 107	\$ 135	\$ 200	\$ 177	\$ 230	
Household	\$ 842	Jun	\$ 282	\$ 1,281	\$ 937	\$ 240	\$ 774	\$ 176	\$ 1,139	\$ 652	\$ 254	\$ 745	
Kid's Komer	\$ 120	Jul	\$ 120	\$ 1,020	\$ 1,020	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	
Travel	\$ 1,763	Aug	\$ 304	\$ 1,032	\$ 176	\$ 398	\$ 356	\$ 430	\$ 190	\$ 241	\$ 407	\$ 259	
Electronics	\$ 301	Sep	\$ 176	\$ 2,958	\$ 587	\$ 340	\$ 1,452	\$ 1,071	\$ 315	\$ 210	\$ 202	\$ 165	
Food	\$ 200	Oct	\$ 200	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	
Furniture	\$ 39	Nov	\$ 1,602	\$ 1,082	\$ 1,187	\$ 842	\$ 745	\$ 145	\$ 101	\$ 1,037	\$ 37		
Gifts	\$ 200	Dec	\$ 200	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	
Health & Beauty	\$ 311	Jan	\$ 174	\$ 2,634	\$ 1,130	\$ 954	\$ 2,083	\$ 1,351	\$ 747	\$ 425	\$ 447	\$ 1,141	
Household	\$ 258	Feb	\$ 702	\$ 1,123	\$ 1,336	\$ 1,227	\$ 3,887	\$ 545	\$ 2,000	\$ 277	\$ 262		
Kid's Komer	\$ 120	Mar	\$ 120	\$ 1,020	\$ 1,020	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	
Travel	\$ 407	Apr	\$ 941	\$ 924	\$ 712	\$ 133	\$ 2,486	\$ 49	\$ 390	\$ 221	\$ 221	\$ 46	
Electronics	\$ 667	May	\$ 1,721	\$ 440	\$ 145	\$ 80	\$ 2,110	\$ 303	\$ 100	\$ 557	\$ 145		
Food	\$ 200	Jun	\$ 200	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	\$ 1,000	
Furniture	\$ 586	Jul	\$ 1,997	\$ 412	\$ 226	\$ 406	\$ 361	\$ 1,626	\$ 267	\$ 1,011	\$ 41	\$ 104	
Gifts	\$ 100	Aug	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	
Health & Beauty	\$ 338	Sep	\$ 179	\$ 655	\$ 427	\$ 99	\$ 2,576	\$ 885	\$ 130	\$ 85	\$ 1,110	\$ 310	
Household	\$ 544	Oct	\$ 413	\$ 1,467	\$ 2,09	\$ 879	\$ 705	\$ 556	\$ 400	\$ 485	\$ 99	\$ 160	
Kid's Komer	\$ 110	Nov	\$ 110	\$ 1,020	\$ 1,020	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	\$ 105	
Travel	\$ 896	Dec	\$ 2,096	\$ 1,726	\$ 3,642	\$ 295	\$ 1,740	\$ 1,943	\$ 366	\$ 307	\$ 318		



- **Drill down:** aumento di dettaglio dei dati:

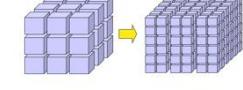
 - Riducendo il livello di gerarchia (cambio delle clausole della group by)
 - Aggiungendo una nuova dimensione (alla group by)

Metrics	Dollar Sales	Customer Region	North-East	Mid-Atlantic	South-East	Central	South	North-West	South-West	England	France	Germany	Canada
Quarter													
Q1 1997	\$ 1,526	1	\$ 1,249	\$ 978	\$ 1,885	\$ 3,650	\$ 4,855	\$ 1,834	\$ 2,552	\$ 1,920	\$ 643	\$ 663	
Q2 1997	\$ 1,599	2	\$ 1,249	\$ 978	\$ 1,885	\$ 3,650	\$ 4,855	\$ 1,834	\$ 2,552	\$ 1,920	\$ 643	\$ 663	
Q3 1997	\$ 3,016	3	\$ 1,772	\$ 5,076	\$ 2,050	\$ 1,443	\$ 2,311	\$ 2,321	\$ 608	\$ 575	\$ 782	\$ 325	
Q4 1997	\$ 2,711	4	\$ 5,030	\$ 2,025	\$ 1,961	\$ 3,601	\$ 2,112	\$ 3,976	\$ 918	\$ 531	\$ 1,676	\$ 291	
Q1 1998	\$ 1,525	5	\$ 1,249	\$ 978	\$ 1,885	\$ 3,650	\$ 4,855	\$ 1,834	\$ 2,552	\$ 1,920	\$ 643	\$ 663	
Q2 1998	\$ 1,773	6	\$ 1,249	\$ 978	\$ 1,885	\$ 3,650	\$ 4,855	\$ 1,834	\$ 2,552	\$ 1,920	\$ 643	\$ 663	
Q3 1998	\$ 1,818	7	\$ 5,402	\$ 1,709	\$ 2,485	\$ 1,704	\$ 3,632	\$ 4,329	\$ 679	\$ 1,198	\$ 1,269	\$ 809	
Q4 1998	\$ 2,051	8	\$ 2,968	\$ 4,664	\$ 5,921	\$ 1,770	\$ 3,162	\$ 3,489	\$ 2,790	\$ 1,005	\$ 846	\$ 639	



- **Slice and dice:** riduzione del volume dei dati da analizzare:

 - **Slice:** predicato di uguaglianza che seleziona una "fetta" (lungo una dimensione di riferimento)
 - **Dice:** combinazione di predicati che seleziona un cubetto (porzione di dimensioni)



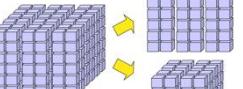
Metrics	Dollar Sales	Customer City	Arlin	San Pedro	Springfield	Chappell Hill	Scorburg	Pebble Beach	Martinsville	Maddon	Peoria	Pecos	Lake Barkley	Alameda	Fingers Lake
Quarter															
Q1 1997	\$ 675														
Q2 1997	\$ 203														
Q3 1997	\$ 215	9	\$ 124		\$ 113	\$ 45	\$ 192	\$ 348							
Q4 1997	\$ 140	10	\$ 174												
Q1 1998	\$ 734														
Q2 1998	\$ 256														
Q3 1998	\$ 219														
Q4 1998	\$ 209														



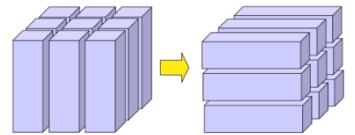
Metrics	Dollar Sales	Customer Region	North-East	Mid-Atlantic	South-East	Central	South	North-West	North-West
Category									
Electronics	\$ 130	1997	\$ 1,774	\$ 384	\$ 138	\$ 2,346	\$ 2,554	\$ 2,148	\$ 566
Food	\$ 1,184	1998	\$ 4,529	\$ 1,052	\$ 1,232	\$ 2,663	\$ 2,462	\$ 1,740	\$ 487
Furniture	\$ 207	1997	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020
Gifts	\$ 538	1998	\$ 955	\$ 677	\$ 213	\$ 1,503	\$ 261	\$ 165	\$ 325
Health & Beauty	\$ 2,711	1997	\$ 1,955	\$ 1,955	\$ 1,955	\$ 2,112	\$ 3,976	\$ 918	\$ 1,156
Household	\$ 1,525	1998	\$ 2,785	\$ 2,800	\$ 1,813	\$ 2,844	\$ 1,778	\$ 1,158	\$ 6
Kid's Komer	\$ 247	1997	\$ 422	\$ 441	\$ 380	\$ 221	\$ 592	\$ 198	\$ 19
Travel	\$ 624	1998	\$ 505	\$ 564	\$ 396	\$ 300	\$ 978	\$ 416	\$ 40



Metrics	Dollar Sales	Customer Region	North-East	Mid-Atlantic	South-East	Central	South	North-West	North-West
Category									
Electronics	\$ 1,184	1997	\$ 4,529	\$ 1,892	\$ 7,232	\$ 651	\$ 9,488	\$ 476	\$ 702
Food	\$ 538	1998	\$ 955	\$ 677	\$ 213	\$ 1,503	\$ 261	\$ 165	\$ 325
Furniture	\$ 1,095	1997	\$ 2,800	\$ 2,695	\$ 1,813	\$ 2,644	\$ 1,776	\$ 1,156	\$ 686
Gifts	\$ 1,123	1998	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020	\$ 1,020
Health & Beauty	\$ 1,842	1997	\$ 2,982	\$ 2,982	\$ 2,982	\$ 2,982	\$ 2,982	\$ 2,982	\$ 2,982
Household	\$ 5,787	1998	\$ 3,320	\$ 5,416	\$ 6,812	\$ 4,334	\$ 5,008	\$ 7,588	\$ 2,149
Kid's Komer	\$ 247	1997	\$ 422	\$ 441	\$ 380	\$ 221	\$ 592	\$ 198	\$ 19
Travel	\$ 609	1998	\$ 559	\$ 1,096	\$ 611	\$ 464	\$ 316	\$ 573	\$ 257



- **Pivot:** riorganizzazione dell'orientamento della struttura multidimensionale senza variare il livello di dettaglio



Category	Metrics	Dollar Sales										
		Year	North-East	Mid-Atlantic	South-East	Central	South	North-West	South-West	England	France	Germany
Electronics	1997	\$ 10.616	\$ 1.184	\$ 1.774	\$ 384	\$ 138	\$ 2.946	\$ 2.554	\$ 2.184	\$ 566	\$ 199	
	1998	\$ 29.299	\$ 4.529	\$ 1.692	\$ 7.232	\$ 651	\$ 4.488	\$ 4.476	\$ 2.683	\$ 462	\$ 7	
Food	1997	\$ 5.300	\$ 562	\$ 729	\$ 262	\$ 598	\$ 469	\$ 807	\$ 156	\$ 615	\$ 1	
	1998	\$ 5.638	\$ 682	\$ 785	\$ 278	\$ 613	\$ 523	\$ 833	\$ 162	\$ 675	\$ 1	
Gifts	1997	\$ 16.315	\$ 1.955	\$ 2.785	\$ 625	\$ 1.143	\$ 2.535	\$ 2.132	\$ 1.904	\$ 908	\$ 375	\$ 1.0
	1998	\$ 20.047	\$ 2.352	\$ 3.955	\$ 2.800	\$ 2.695	\$ 1.813	\$ 2.844	\$ 1.778	\$ 1.158	\$ 717	\$ 6
Health & Beauty	1997	\$ 4.402	\$ 464	\$ 645	\$ 1.413	\$ 568	\$ 1.044	\$ 1.044	\$ 702	\$ 273	\$ 5	
	1998	\$ 5.665	\$ 611	\$ 887	\$ 566	\$ 382	\$ 499	\$ 1.162	\$ 1.044	\$ 273	\$ 72	
Household	1997	\$ 38.383	\$ 3.554	\$ 4.112	\$ 5.410	\$ 4.446	\$ 3.058	\$ 3.974	\$ 2.654	\$ 2.945	\$ 2.875	\$ 1.9
	1998	\$ 40.241	\$ 3.554	\$ 4.112	\$ 5.410	\$ 4.446	\$ 3.058	\$ 3.974	\$ 2.654	\$ 2.945	\$ 2.875	\$ 1.9
Kid's Korner	1997	\$ 2.559	\$ 201	\$ 398	\$ 485	\$ 186	\$ 409	\$ 323	\$ 396	\$ 105	\$ 34	
	1998	\$ 2.943	\$ 247	\$ 422	\$ 441	\$ 380	\$ 221	\$ 592	\$ 290	\$ 198	\$ 19	
Travel	1997	\$ 4.497	\$ 624	\$ 595	\$ 464	\$ 380	\$ 300	\$ 978	\$ 416	\$ 146	\$ 38	
	1998	\$ 4.792	\$ 608	\$ 559	\$ 1.096	\$ 611	\$ 464	\$ 316	\$ 978	\$ 257	\$ 198	

Estensione del linguaggio SQL

Per l'analisi mediante strumenti OLAP c'è la necessità di estendere il linguaggio SQL con nuove funzioni di aggregazione e per la generazione di rapporti.

Le nuove funzioni OLAP includono:

- Finestra di calcolo: in cui è possibile specificare il calcolo di aggregati mobili e totali cumulativi su un gruppo di record , inoltre permette il confronto fra dati dettagliati e dati complessivi (percentuale vendite di un negozio rispetto a tutti i negozi)
- Funzioni di ranking: consentono di ricavare la posizione nell'ordinamento

La finestra di calcolo è caratterizzata da diverse clausole:

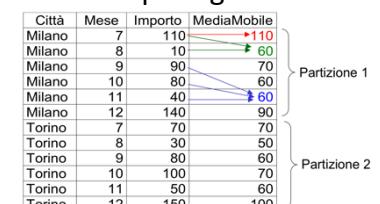
- **Partizionamento (PARTITION BY):** divide le righe in gruppi senza collassarle come farebbe la group by, in assenza di partizionamento ci sarà un solo gruppo
- **Ordinamento delle righe (ORDER BY):** le righe vengono ordinate separatamente all'interno di ogni partizione
- **Finestra di aggregazione (ROWS/RANGE):** definisce il gruppo di righe su cui l'aggregato è calcolato, per ciascuna riga della partizione

Esempio: Visualizzare, per ogni città e mese:

- L'importo delle vendite
- La media rispetto al mese corrente e ai due mesi precedenti, separatamente per ogni città

```
SELECT Città, Mese, Importo,
       AVG(Importo) OVER Wavg AS MediaMobile
    FROM Vendite
   WINDOW Wavg AS (PARTITION BY Città
                    ORDER BY Mese
                    ROWS 2 PRECEDING)
```

```
SELECT Città, Mese, Importo,
       AVG(Importo) OVER (PARTITION BY Città
                           ORDER BY Mese
                           ROWS 2 PRECEDING)
      AS MediaMobile
    FROM Vendite
```



In questo caso è necessario specificare l'**ordinamento**, in quanto l'aggregazione richiede utilizza le righe in modo ordinato, si nota anche che quando la finestra è **incompleta** il calcolo viene effettuato sulla parte della finestra presente (è possibile specificare che in tal caso il risultato deve essere NULL).

Si possono avere più finestre di calcolo diverse nella stessa query.

La finestra di aggregazione può essere definita:

- **A livello fisico:** formando il gruppo mediante conteggio delle righe, è adatto per dati più densi
 - ROWS 2 PRECEDING (o FOLLOWING)
 - ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING
 - ROWS UNBOUNDED PRECEDING
- **A livello logico:** formando il gruppo in base alla definizione di un intervallo intorno alla chiave di ordinamento, è adatto per dati sparsi
 - RANGE 2 MONTH PRECEDING

Esempio (totali cumulativi): Visualizzare, per ogni città e mese

- L'importo delle vendite
- L'importo cumulativo delle vendite al trascorrere dei mesi, separatamente per ogni città

```
SELECT Città, Mese, Importo,
       SUM(Importo) OVER (PARTITION BY Città
                           ORDER BY Mese
                           ROWS UNBOUNDED PRECEDING)
          AS SommaCumul
FROM Vendite
```

Città	Mese	Importo	SommaCumul
Milano	7	110	110
Milano	8	10	120
Milano	9	90	210
Milano	10	80	290
Milano	11	40	330
Milano	12	140	470
Torino	7	70	70
Torino	8	30	100
Torino	9	80	180
Torino	10	100	280
Torino	11	50	330
Torino	12	150	480

Esempio (confronto fra dati dettagliati e dati complessivi): Visualizzare, per ogni città e mese

- L'importo delle vendite
- L'importo totale delle vendite sul periodo completo per la città corrente

```
SELECT Città, Mese, Importo,
       SUM(Importo) OVER (PARTITION BY Città)
          AS ImpTotale
FROM Vendite
```

Città	Mese	Importo	ImpTotale
Milano	7	110	470
Milano	8	10	470
Milano	9	90	470
Milano	10	80	470
Milano	11	40	470
Milano	12	140	470
Torino	7	70	480
Torino	8	30	480
Torino	9	80	480
Torino	10	100	480
Torino	11	50	480
Torino	12	150	480

Esempio (confronto fra dati dettagliati e dati complessivi): Visualizzare, per ogni città e mese

- L'importo delle vendite
- Il rapporto tra l'importo della riga corrente per le vendite e il totale complessivo
- Il rapporto tra l'importo della riga corrente per le vendite e il totale complessivo per città
- Il rapporto tra l'importo della riga corrente per le vendite e il totale complessivo per mese

Nella prima finestra di calcolo è necessario notare che l'inserimento di OVER () è fondamentale in quanto non è possibile eseguire un'operazione fra un valore e un aggregato in SQL non esteso.

```
SELECT Città, Mese, Importo
       Importo/SUM(Importo) OVER ()
          AS PercTotale
       Importo/SUM(Importo) OVER (PARTITION BY Città)
          AS PercCittà
       Importo/SUM(Importo) OVER (PARTITION BY Mese)
          AS PercMese
FROM Vendite
```

Città	Mese	Importo	PercTotale	PercCittà	PercMese
Milano	7	110	110/950	110/470	110/180
Milano	8	10	10/950	10/470	10/40
Milano	9	90	90/950	90/470	90/170
Milano	10	80	80/950	80/470	80/180
Milano	11	40	40/950	40/470	40/90
Milano	12	140	140/950	140/470	140/290
Torino	7	70	70/950	70/480	70/180
Torino	8	30	30/950	30/480	30/40
Torino	9	80	80/950	80/480	80/170
Torino	10	100	100/950	100/480	100/180
Torino	11	50	50/950	50/480	50/90
Torino	12	150	150/950	150/480	150/290

È possibile **abbinare** l'uso di **finestre di calcolo** con il raggruppamento eseguito dalla clausola **group by**, la tabella generata diviene l'operando a cui applicare le operazioni definite per la finestra di calcolo.

Importante!

All'interno di una finestra di calcolo è possibile usare solo le colonne disponibili a livello di SELECT (non necessariamente visualizzate): ad esempio nella query

```
SELECT Città, Mese, SUM(Importo) AS TotMese,
       AVG(SUM(Importo)) OVER (PARTITION BY Città)
          AS MediaMobile
FROM Vendite, ...
WHERE <cond. join> GROUP BY Città, Mese
```

non è possibile utilizzare, nel contesto della finestra di calcolo, l'attributo Importo, ma solo SUM(Importo), e non è possibile neanche calcolare AVG(Importo) come operazione relativa alla finestra di calcolo perché Importo non è disponibile a livello di SELECT.

Le funzioni di ranking servono per calcolare la posizione di un valore all'interno di una partizione:

- **RANK ()**: calcola la posizione, lasciando intervalli vuoti successivi in presenza di record a pari merito (1,1,3)
- **DENSERANK ()**: calcola la posizione senza lasciare intervalli vuoti successivi alla presenza di pari merito (1,1,2)

In entrambi i casi è possibile computare i top n record del ranking facendo WHERE Rank<=n

Esempio (ranking): Visualizzare, per ogni città nel mese di dicembre

- l'importo delle vendite
 - la posizione nella graduatoria
- ```
SELECT Città, Importo,
 RANK() OVER (ORDER BY Importo DESC)
 AS Graduatoria
 FROM Vendite
 WHERE Mese = 12
```

Il risultato non è ordinato di default

| Città  | Importo | Graduatoria |
|--------|---------|-------------|
| Torino | 150     | 1           |
| Milano | 140     | 2           |

in base all'ordine in graduatoria, lo si può fare mediante un ORDER BY

La clausola GROUP BY viene esteso con tre diverse possibilità:

- **Rollup**: consente di calcolare le aggregazioni su tutti i gruppi ottenuti togliendo una colonna per volta dall'insieme specificato di colonne  
**Esempio**: GROUP BY ROLLUP a,b,c => group by a,b,c; GROUP BY a,b ; GROUP BY a ; no GROUP BY
- **Cube**: consente di calcolare le aggregazioni su tutte le possibili combinazioni delle colonne specificate
- **Grouping sets**: consente di specificare un elenco di raggruppamenti richiesti  
**Esempio**: GROUP BY GROUPING SET (a,(a,b,c)) => GROUP BY a; GROUP BY a,b,c

Per ognuna di queste clausole si verranno a creare dei risultati simili alla tabella affianco dove quando troviamo un NULL, significa che quel record non è aggregato rispetto a quella colonna, ma solo per le altre valorizzate.

| Città  | Mese | Pkey | TotVendite |
|--------|------|------|------------|
| Milano | 7    | 145  | 110        |
| Milano | 7    | 150  | 10         |
| Milano | ...  | ...  | ...        |
| Milano | 7    | NULL | 8500       |
| Milano | 8    | ...  | ...        |
| Milano | NULL | NULL | 150000     |
| Torino | ...  | ...  | 150        |
| Torino | ...  | NULL | 2500       |
| Torino | NULL | NULL | 135000     |
| ...    | ...  | ...  | ...        |
| NULL   | NULL | NULL | 25005000   |

La funzione **ROW\_NUMBER** assegna un numero progressivo ad ogni riga all'interno di una partizione.

Esempio:

```
SELECT Tipo, Peso, ROW_NUMBER OVER (PARTITION BY Tipo ORDER BY Peso)
 AS RowNumberPeso FROM ART;
```

| Tipo        | Peso | RowNumberPeso |              |
|-------------|------|---------------|--------------|
| Barra       | 12   | 1             | Partizione 1 |
| Ingranaggio | 19   | 1             | Partizione 2 |
| Vite        | 12   | 1             | Partizione 3 |
| Vite        | 14   | 2             |              |
| Vite        | 16   | 3             |              |
| Vite        | 16   | 4             |              |
| Vite        | 16   | 5             |              |
| Vite        | 16   | 6             |              |
| Vite        | 17   | 7             |              |
| Vite        | 17   | 8             |              |
| Vite        | 18   | 9             |              |
| Vite        | 20   | 10            |              |

La funzione **CUME\_DIST** assegna un peso compreso fra 0 e 1 ad ogni riga in funzione del numero di valori distinti che precedono il valore assunto dal campo usato per effettuare l'ordinamento all'interno delle partizioni.

Esempio:

```
SELECT Tipo, Peso, CUME_DIST() OVER (PARTITION BY Tipo
 ORDER BY Peso
) AS CumePeso FROM ART;
```

| Tipo        | Peso | CumePeso    |              |
|-------------|------|-------------|--------------|
| Barra       | 12   | 1 (= 1/1)   | Partizione 1 |
| Ingranaggio | 19   | 1 (= 1/1)   | Partizione 2 |
| Vite        | 12   | .1 (= 1/10) | Partizione 3 |
| Vite        | 14   | .2 (= 2/10) |              |
| Vite        | 16   | .6 (= 6/10) |              |
| Vite        | 16   | .6 (= 6/10) |              |
| Vite        | 16   | .6 (= 6/10) |              |
| Vite        | 17   | .8 (= 8/10) |              |
| Vite        | 17   | .8 (= 8/10) |              |
| Vite        | 18   | .9 (= 9/10) |              |
| Vite        | 20   | 1 (= 10/10) |              |

La funzione **NTILE(n)** divide ogni partizione in n sottogruppi ognuno con lo stesso numero di dati/record (se possibile). Ad ogni sottogruppo viene associato un numero identificativo.

Esempio:

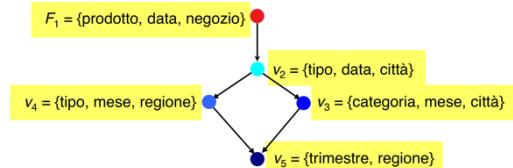
```
SELECT Tipo, Peso, NTILE(3) OVER (PARTITION BY Tipo ORDER BY Peso) AS Ntile3Peso FROM ART;
```

| Tipo        | Peso | Ntile3Peso |               |
|-------------|------|------------|---------------|
| Barra       | 12   | 1          | Partizione 1  |
| Ingranaggio | 19   | 1          | Partizione 2  |
| Vite        | 12   | 1          | Partizione 3  |
| Vite        | 14   | 1          |               |
| Vite        | 16   | 1          | Sottogruppo 1 |
| Vite        | 16   | 1          |               |
| Vite        | 16   | 2          | Sottogruppo 2 |
| Vite        | 16   | 2          |               |
| Vite        | 17   | 2          |               |
| Vite        | 17   | 3          | Sottogruppo 3 |
| Vite        | 18   | 3          |               |
| Vite        | 20   | 3          |               |

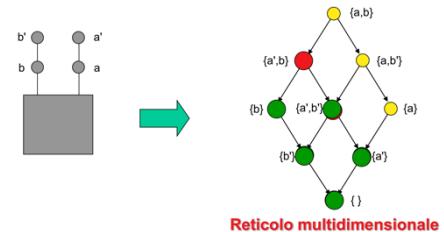
## 5. Viste materializzate

Le **viste materializzate** sono sommari precalcolati della tabella dei fatti che sono **memorizzati** direttamente nel DW e che permettono di aumentare l'**efficienza** delle interrogazioni che richiedono aggregazioni.

Il **reticolo delle viste** rappresenta l'insieme delle viste materializzate di cui si ha bisogno, rappresentando, spostandosi verso il basso, le viste con un contenuto più aggregato.



Le viste materializzate vengono definite da query SQL a partire dalle tabelle sul DW o da viste con un contenuto meno aggregato di quello che si vuole generare. È opportuno inserire anche delle generalizzazioni rispetto alle dimensioni (ad esempio Stato per la dimensione Città) che permettono di riutilizzare la vista pur non occupando molto spazio aggiuntivo.



Una vista materializzata può essere utilizzata per rispondere a più interrogazioni diverse, per questo motivo è necessario identificare l'insieme "ottimo" di viste da realizzare rispetto alle query che verranno eseguite.

È necessario che l'insieme "ottimo" minimizzi i tempi di esecuzione delle query e di aggiornamento delle viste rispettando i vincoli:

- Spazio disponibile su disco
- Tempo a disposizione per l'aggiornamento (solitamente la periodicità dell'aggiornamento è giornaliera)
- Tempo di risposta
- Freschezza dei dati

## Progettazione fisica

La **progettazione fisica** è composta da diverse fasi:

- **Analisi del carico di lavoro:** caratterizzazione delle query, della loro frequenza di esecuzione e della frequenza di aggiornamento dei dati con eventuale ricostruzione di strutture fisiche
- **Definizione delle strutture fisiche accessorie:**
  - Indici:
    - **Indici di bitmap:** permettono di eseguire operazioni legate ai predicati di selezione più velocemente
    - **Indici di join:** permettono di eseguire operazioni di join più velocemente memorizzando delle tabelle che contengono già i row id da andare ad inserire nel join

- **Considerare le caratteristiche dell'ottimizzatore:** che sceglie se e quando utilizzare le viste progettate dall'operatore
  - **Procedimento di progettazione fisica:**
    - Selezione delle strutture adatte per supportare le interrogazioni più frequenti
    - Scelta delle strutture in grado di contribuire al miglioramento di più interrogazioni contemporaneamente
    - Rispetto dei vincoli relativi allo spazio e alla frequenza di aggiornamento dei dati
  - **Tuning:** verificare se la progettazione fisica soddisfa le necessità applicative, se le necessità non vengono soddisfatte le strutture accessorie vanno riprogettate.
- In questa fase si osserva se le strutture fisiche vengono utilizzate o meno dall'ottimizzatore.

### *Scelta degli indici*

Per l'indicizzazione delle **dimensioni** è opportuno creare un indice per gli attributi coinvolti in prediciati di selezione:

- Se il dominio ha cardinalità **elevata** è opportuno utilizzare un indice **B-tree**
- Se il dominio ha cardinalità **ridotta** è opportuno utilizzare un indice **bitmap**

Per l'indicizzazione delle operazioni di **join** è consigliato l'utilizzo di **bitmapped join index** se disponibile.

Per l'indicizzazione delle operazioni di **group by** è opportuno l'uso delle **viste materializzate**.

### *Alimentazione del DW*

Il processo di **Extraction, Transformation and Loading (ETL)** è il processo di preparazione dei dati da introdurre nel DW che è composto da estrazione dei dati dalle sorgenti, pulitura, trasformazione e caricamento.

Questo processo viene eseguito sia durante **il primo popolamento** del DW comunicando prima di tutto le dimensioni, e poi la tabella dei fatti sia durante **l'aggiornamento periodico** dei dati.

### *Estrazione*

Durante la fase di **estrazione** avviene l'acquisizione dei dati dalle sorgenti, può avvenire secondo due diverse modalità:

- **Statica:** viene eseguita una fotografia dei dati dalle sorgenti, questa modalità viene utilizzata per il primo popolamento del DW
- **Incrementale:** vengono presi in considerazione solo i dati nuovi o quelli aggiornati rispetto all'ultima estrazione, questa modalità viene utilizzata per l'aggiornamento periodico del DW

In questa fase è importante avere a disposizione ed estrarre dati di **qualità**.



L'estrazione può avvenire su **dati operazionali di natura diversa**:

- **Storicizzati:** ogni operazione o modifica è memorizzata in maniera puntuale e per un periodo definito di tempo all'interno del sistema OLTP
- **Semi-storicizzati:** nel sistema OLTP viene memorizzato solo un numero limitato di stati, l'estrazione incrementale è più complessa da effettuare
- **Transitori:** il sistema OLTP mantiene solo l'immagine corrente dei dati, l'estrazione diventa operativamente molto complessa

L'**estrazione incrementale** può avvenire mediante:

- L'assistenza di un **applicazione** che cattura le modifiche
- L'uso di **log** che storicizzano le modifiche
- La definizione di **trigger** che catturano le modifiche di interesse
- L'utilizzo di **timestamp** che marcano l'ultima modifica nel contesto di un record, richiede la modifica dello schema della base di dati e può perdere stati intermedi di modifica

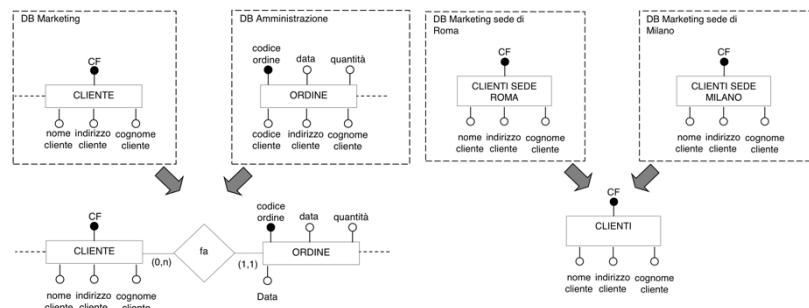
## Pulitura

Durante la fase di **pulitura** vengono eseguite operazioni volte al miglioramento della qualità dei dati, si opera su:

- Dati duplicati
- Dati mancanti
- Uso non previsto di un campo
- Valori impossibili o errati
- Inconsistenza fra valori logicamente associati

Questi problemi possono essere dovuti ad errori di battitura, differenze di formato dei campi o all'evoluzione del modo di operare dell'azienda e ognuno di essi richiede una specifica soluzione:

- **Tecniche basate su dizionari**: adatte per attributi con un dominio ristretto che presentano errori di battitura o formato
- **Tecniche di fusione approssimata**: adatte per il riconoscimento di duplicati/correlazioni tra dati simili:
  - **Join approssimato**: viene fatta una join utilizzando gli attributi comuni che non rappresentano un identificatore (chiave)
  - **Purge/merge**: i dati presenti in due diverse tabelle di partenza vengono uniti in una sola tabella, necessita l'identificazione e l'eliminazione dei dati duplicati e la gestione di due attributi simili ma in un formato diverso
- **Identificazione di outliers** o deviazioni mediante business rules

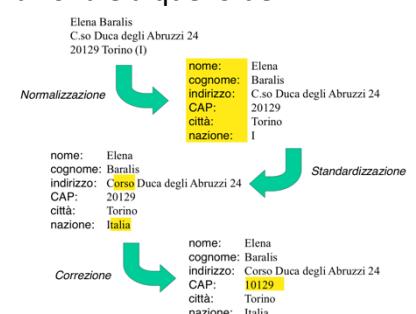


La strategia migliore è la prevenzione, rendendo più affidabili e rigorose le procedure di data entry OLTP.

## Trasformazione

Durante la fase di **trasformazione** i dati vengono convertiti dal formato operazionale a quello del DW, ciò richiede una rappresentazione uniforme del dato che avviene in due fasi:

- Sorgenti operazionali => Dati riconciliati (nella staging area):
  - Conversioni e normalizzazione
  - Matching
  - Eventuale filtraggio dei dati significativi
- Dati riconciliati => DW:
  - Generazione di chiavi surrogate e di valori aggregati



## Caricamento

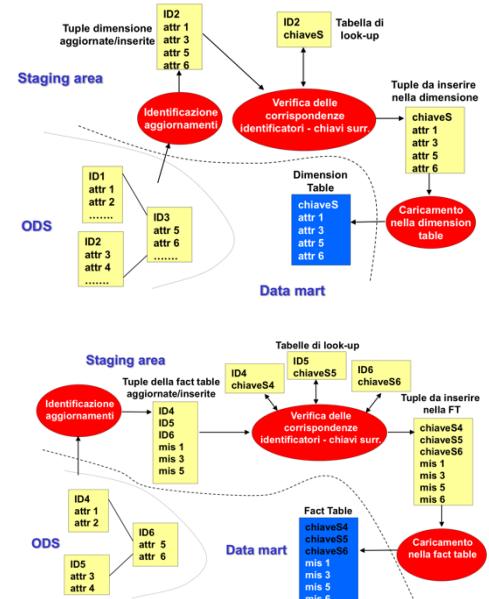
Durante la fase di **caricamento** gli aggiornamenti vengono propagati al DW, per mantenere l'integrità le modifiche vengono effettuate nel seguente ordine:

1. Dimensioni
2. Tabella dei fatti
3. Viste materializzate

### Ripercorrendo l'alimentazione delle tabelle delle dimensioni

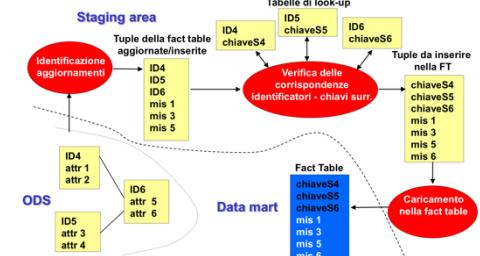
abbiamo:

- L'identificazione degli aggiornamenti
- L'individuazione della chiave surrogata mediante una tabella di lookup
- L'aggiornamento dei dati nella tabella delle dimensioni



### Ripercorrendo invece l'alimentazione delle fact tables

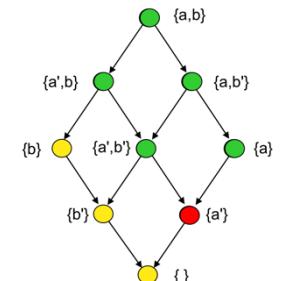
- abbiamo:
- L'identificazione degli aggiornamenti
  - L'individuazione delle chiavi surrogate relative alle dimensioni mediante tabelle di look-up
  - Inserimento delle misure e caricamento nelle tabelle dei fatti



**L'alimentazione delle viste materializzate** invece può essere fatto in due modi:

- A partire direttamente dalla **tabella dei fatti** se l'estrazione è di tipo statico
- Dalle **sorgenti** se l'estrazione è di tipo incrementale

Le viste più in basso si aggiornano a partire da quelle meno aggregate (più in alto).



Una vista materializzata può essere utilizzata in qualunque interrogazione come se fosse una tabella.

È possibile creare una vista materializzata mediante:

```
CREATE MATERIALIZED VIEW Name
[BUILD {IMMEDIATE | DEFERRED}]
[REFRESH {COMPLETE | FAST | FORCE | NEVER} {ON COMMIT | ON DEMAND}]
[ENABLE QUERY REWRITE]
```

AS Query:

- **Name:** nome della vista
- **Query:** interrogazione associata alla vista
- **BUILD**
  - **IMMEDIATE:** crea la vista materializzata e carica immediatamente i risultati dell'interrogazione al suo interno
  - **DEFERRED:** crea la vista materializzata ma non carica i dati associati all'interrogazione al suo interno

- **REFRESH**
  - **COMPLETE**: ricalcola il risultato dell'interrogazione eseguendola su tutti i dati
  - **FAST**: aggiorna il contenuto della vista materializzata basandosi sulle variazioni avvenute dall'ultima operazione di refresh, utilizza dei file di log da cui computa quali dati sono da aggiornare
  - **FORCE**: se possibile viene eseguito il refresh fast, altrimenti viene usata la modalità COMPLETE
  - **NEVER**: il contenuto della vista non viene aggiornato con le procedure standard Oracle
  - **ON COMMIT**: il refresh viene effettuato automaticamente quando le operazioni SQL eseguite comportano una variazione del contenuto della vista materializzata
  - **ON DEMAND**: il refresh viene effettuato solo su richiesta esplicita dell'utente usando la procedura **DBMS\_MVIEW.REFRESH('vista','C'|'F')** specificando il nome della vista da aggiornare e se si vuole fare un refresh di tipo COMPLETE o FORCE
- **ENABLE QUERY REWRITE**: abilita il DBMS ad utilizzare la vista materializzata come blocco base per eseguire più "velocemente" altre interrogazioni

Esempio: Voglio "materializzare" l'interrogazione:

```
SELECT Cod_F, Cod_A, SUM(Q) FROM FAP GROUP BY Cod_F, Cod_A
```

Con caricamento dei dati immediato, refresh completo operato solo su richiesta dell'utente e abilitazione alla riscrittura delle interrogazioni

```
CREATE MATERIALIZED VIEW Frn_Art_sumQ
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE
AS SELECT Cod_F, Cod_A, SUM(Q) FROM FAP GROUP BY Cod_F, Cod_A;
```

I **file di log**, richiesti obbligatoriamente per l'esecuzione di un **FAST REFRESH**, sono collegati alla singola tabella e contengono le variazioni della stessa, o di un gruppo di suoi attributi.

Esempio: Creare un materialized view log associato alla tabella FAP e in particolare agli attributi Cod\_F, Cod\_A, Q abilitando le opzioni SEQUENCE e ROWID e la gestione di nuovi valori

```
CREATE MATERIALIZED VIEW LOG ON FAP
WITH SEQUENCE, ROWID
(Cod_F, Cod_A, Q)
INCLUDING NEW VALUES;
```

In cui **SEQUENCE** identifica l'istante in cui è avvenuta la variazione e **ROWID** la tupla modificata, i campi specificati successivamente rappresentano gli attributi di cui il file di log deve tener traccia e **INCLUDING NEW VALUES** indica la gestione di nuovi valori sugli attributi.

È possibile eliminare una vista con **DROP MATERIALIZED VIEW Nome** o modificarne una mediante **ALTER MATERIALIZED VIEW Nome opzioni**.

## 6. Data lakes

I **data lakes** sono repository di dati di tipo **raw** (grezzi) e semplicemente storicizzati senza nessuna particolare elaborazione da cui è possibile estrarre **valore** se appositamente lavorati.

Possono contenere dati di diversa natura:

- Dati strutturati (relazionali)
- Dati semi-strutturati (csv, json, xml)
- Dati non strutturati (testo)
- Dati binari (immagini, audio)

In questo tipo di sistemi le operazioni di ricerca di un dato sono più complesse e più simili ad una ricerca su Google in cui però c'è bisogno di effettuare operazioni di **data wrangling** per convertire i dati in un formato più fruibile.

Quando si sceglie di implementare un data lake non si sa a priori quale valore e quale utilizzo avranno i dati memorizzati, permette inoltre di inserire tutti dati in un unico luogo, non lasciandoli sparsi in sistemi di memorizzazione distribuiti.

Un data lake:

- Memorizza tutti i dati, anche quelli che non si è sicuri possano diventare utili in futuro
- Gestisce dati di qualsiasi tipo
- È utile ad altri tipi di utenti e non solo agli attori di business (ad esempio ai data scientist)
- Si adatta facilmente ai cambiamenti
- Fornisce informazioni utili più velocemente

| Data warehouse                                   | Data lake                                                                                                                           |
|--------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Contiene dati <b>relazionali</b>                 | Può anche contenere dati <b>non-relazionali</b>                                                                                     |
| Lo <b>schema</b> del DW è definito a priori      | Lo <b>schema</b> viene scritto durante l'analisi                                                                                    |
| Alti <b>costi</b>                                | Bassi costi                                                                                                                         |
| Dati di <b>qualità</b>                           | Nessuna certezza sulla <b>qualità</b>                                                                                               |
| Utilizzato dai business analysts                 | Utilizzato dai data scientist e dai data developers                                                                                 |
| Analytics: BI e visualizzazione, batch reporting | Analytics: full-text search, machine learning, predictive analytics, data discovery, profiling e utilizzo di algoritmi unsupervised |

Pro:

- Raccolta di più dati, da più sorgenti, in meno tempo
- Le strutture dati e i requisiti sono definiti solo quando necessari
- Permette l'esplorazione dei dati
- L'integrazione avviene fuori dallo storage
- Coinvolgimento minimo dell'area IT

Contro:

- I dati sono memorizzati senza essere sicuri del contenuto
- La consistenza e la qualità dei dati non sono certe

- I dati non sono utilizzabili da business users
- Query ostiche possono rallentare grandi cluster

Inizialmente veniva storicozzato, all'interno dei data lake, qualsiasi dato a disposizione. Oggi giorno data la grossa mole di informazioni memorizzabili si sceglie di avere un approccio più conservativo sui dati da storicozzare per non rischiare di avere grandi e inutili insiemi di dati.

## 8. Data preprocessing

### Dati

I dati sono una collezione di **oggetti** e i loro **attributi**.

Gli **attributi** sono una proprietà o una caratteristica dell'oggetto, un insieme di attributi descrive un **oggetto**.

Gli attributi possono essere di vari tipi:

- **Nominali** (ID, colore degli occhi, codice zip)
- **Ordinali** (numeri interi come in un ranking)
- **Intervalli**
- **Rapporti** (valori FP)

Il tipo di un attributo dipende da quali delle seguenti proprietà possiede:

- **Distingibilità** (==, !=) -> Tutti i tipi
- **Ordine** (<, >) -> Ordinale, Intervallo e Rapporto
- **Addizione** (+, -) -> Intervallo e Rapporto
- **Moltiplicazione** (\*, /) -> Rapporto

Un attributo può essere:

- **Discreto**: ha un numero finito, oppure infinito numerabile di valori possibili. Di solito è rappresentato da variabili intere.  
Gli attributi **binari** sono un caso particolare di attributi discreti.
- **Continuo**: consiste in un valore reale che quindi ha un numero infinito non numerabile di valori possibili. Sono solitamente rappresentati da variabili FP

Identificare il tipo di dato serve a mappare il dato al problema e identificare se serve fare preprocessing.

### Data set

Un **data set** può essere di diversi tipi:

- **Record**:
  - **Dati di tipo tabellare**: consistono in una collezione di record caratterizzati da un numero fisso di attributi, il nome della colonna è definito una volta sola, alla creazione della tabella.
  - **Dati di documenti**: include dati testuali che possono essere semi-strutturati o non strutturati, potremo avere un record per ogni frase/paragrafo.

| Tid | Refund | Marital Status | Taxable Income | Cheat |
|-----|--------|----------------|----------------|-------|
| 1   | Yes    | Single         | 125K           | No    |
| 2   | No     | Married        | 100K           | No    |
| 3   | No     | Single         | 70K            | No    |
| 4   | Yes    | Married        | 120K           | No    |
| 5   | No     | Divorced       | 95K            | Yes   |
| 6   | No     | Married        | 60K            | No    |
| 7   | Yes    | Divorced       | 220K           | No    |
| 8   | No     | Single         | 85K            | Yes   |
| 9   | No     | Married        | 75K            | No    |
| 10  | No     | Single         | 90K            | Yes   |

|            | Term | cond | y | phi | beta | slope | gamma | alpha | lambda | beta0 | lambda0 | sigma0 |
|------------|------|------|---|-----|------|-------|-------|-------|--------|-------|---------|--------|
| Document 1 | 3    | 0    | 5 | 0   | 2    | 6     | 0     | 2     | 0      | 2     | 0       | 2      |
| Document 2 | 0    | 7    | 0 | 2   | 1    | 0     | 0     | 0     | 3      | 0     | 0       | 0      |
| Document 3 | 0    | 1    | 0 | 0   | 1    | 2     | 2     | 0     | 3      | 0     | 0       | 0      |

Il testo acquisito in diversi contesti potrebbe avere una specifica struttura o semantica (le pagine web hanno i tag, altri documenti potrebbero avere dei metadata o delle parti evidenziate)

- **Dati transazionali:** ogni record contiene un ID e un insieme di oggetti, sia gli oggetti che le transazioni non sono ordinate.

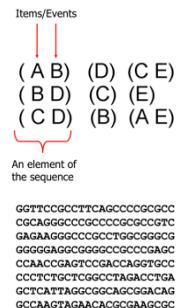
- **Grafo:** insieme di nodi e archi con un determinato peso.

Ad esempio potremmo avere una pagina come un nodo e l'arco può corrispondere ad un link.

- **Ordinato:**

- **Sequenza di transazioni:** ogni transazione è formata da una sequenza di oggetti, in cui ogni oggetto è separato dalle parentesi tonde, non ci interessa l'ordine all'interno dell'oggetto ma l'ordine degli oggetti all'interno della sequenza.
- **Sequenza di geni:** esempio in cui la sequenza non è per forza legata ad un ordine temporale.
- **Sequenze spazio-temporali:** mappa delle temperature medie mensili in diverse regioni geografiche.

| TID | Items                     |
|-----|---------------------------|
| 1   | Bread, Coke, Milk         |
| 2   | Beer, Bread               |
| 3   | Beer, Coke, Diaper, Milk  |
| 4   | Beer, Bread, Diaper, Milk |
| 5   | Coke, Diaper, Milk        |



Una scarsa **qualità dei dati** può essere causa di maggiori sforzi in fase di data preprocessing.

Bisogna conoscere, trovare e limitare i problemi relativi alla qualità dei dati che possono essere:

- **Rumore:** si riferisce a dati che hanno subito una distorsione.
  - **Outliers:** sono oggetti nel dataset con caratteristiche considerevolmente differenti dal resto del dataset. Possono essere sia un dato da escludere, sia l'obiettivo della nostra analisi.
  - **Valori mancanti:** alcuni dati possono mancare per via di informazioni non raccolte sia a causa del fatto che alcuni attributi non sono utilizzabili in tutti i casi.
- È possibile gestire questa situazione in vari modi:
- Eliminare gli oggetti con dati mancanti (potremmo andare a rimuovere una grande fetta di dati)
  - Stimare i valori mancanti in base a valori precedenti/successivi
  - Ignorare i valori mancanti
- **Dati duplicati:** possono essere duplicati a causa di una data integration, nell'unione dei dati di due db che presentano lo stesso record. Nella gestione di questa problematica bisogna assicurarsi che il dato sia effettivamente duplicato e non una reiterazione di una azione.
  - **Dati errati**

## Tecniche di data preprocessing

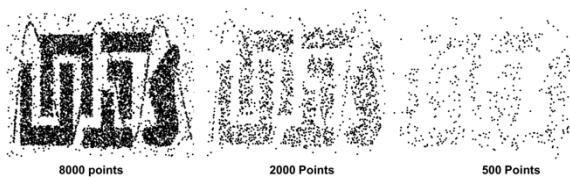
Il **data preprocessing** ha diverse tecniche:

- **Aggregazione:** combinazione di due o più attributi o oggetti in un singolo attributo o oggetto.

Può avere diversi scopi:

- **Riduzione dei dati:** riduzione del numero di attributi o oggetti
- **Cambio di scala:** città aggregate in regioni, stati, continenti
- **Dati più "stabili":** i dati aggregati tendono ad avere meno variabilità

- **Sampling:** vengono presi in considerazione solo alcuni oggetti del set che rappresentino un campione rappresentativo dello stesso (ne ha le stesse caratteristiche). Viene spesso impiegato perché ottenere o processare l'intero dataset è troppo costoso.



Ci possono essere varie tipologie di sampling:

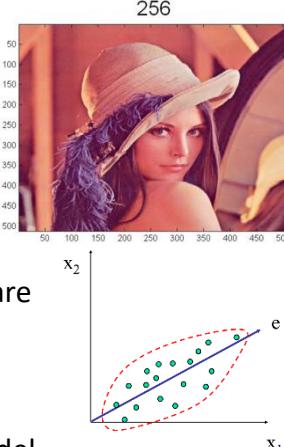
- **Sampling casuale:** c'è la stessa possibilità di selezionare qualsiasi item
  - **Senza rimpiazzo:** un elemento selezionato viene rimosso dalla popolazione
  - **Con rimpiazzo:** un elemento selezionato rimane nella popolazione e può essere rielezionato successivamente
- **Sampling stratificato:** i dati vengono divisi in diverse partizioni, in seguito viene selezionata una percentuale di samples da ogni partizione

La **curse of dimensionality** rappresenta il fatto che all'aumentare delle dimensioni, i dati diventa più sparsi e serve un numero di sample maggiore per effettuare analisi su di essi, per risolvere questo problema può essere utile la riduzione dimensionale.

- **Riduzione dimensionale:** ha l'obiettivo di facilitare la visualizzazione del dato non considerando le features meno significative. Ciò consente di evitare la curse of dimensionality, ridurre i tempi e la memoria richiesti per il data mining.

Le tecniche di riduzione dimensionale sono:

- **Principal Component Analysis (PCA):** l'obiettivo è quello di trovare una proiezione degli attributi che capti la maggiore variazione tra i dati, gli attributi ottenuti non sono uguali e sono una combinazione lineare di quelli originali
- Singular Value Decomposition
- Altre tecniche supervised e non lineari
- **Selezione di un subset di feature:** (è un altro modo di ridurre la dimensionalità del dato) selezioniamo solo gli attributi significativi già presenti, senza crearne di nuovi. Vengono eliminati gli attributi ridondanti e quelli non rilevanti.



Le tecniche principali sono:

- **Brute-force:** (può essere usato in qualsiasi punto della pipeline di analisi) vengono provate tutte le possibilità come input per l'algoritmo di data science
- **Embedded:** la selezione avviene come parte dell'algoritmo di data science, è conveniente utilizzare questo approccio solo in presenza di pochi attributi, se no l'algoritmo impiegherebbe troppo tempo a convergere
- **Filter:** la selezione avviene prima che l'algoritmo di data science venga eseguito, ad esempio possiamo utilizzare l'approccio di Pearson per calcolare la correlazione fra le variabili, se abbiamo un indice **maggiore di 0,8** consideriamo le variabili come correlate
- **Wrapper:** l'algoritmo di data science viene utilizzato come black-box per trovare il subset migliore. Il modello identifica combinazioni diverse di attributi dando diversi output. Mediante il confronto degli output, seleziono quello che genera un risultato più accurato.

Differentemente dal brute-force viene analizzato solo l'output di un gruppo di possibili combinazioni e non di tutte le combinazioni.

- **Creazione di feautures:** consiste nella creazione di un nuovo attributo che catturi informazioni importanti in maniera più efficiente a partire dagli attributi originali.

Permette anche di mappare il dato in uno spazio diverso (tempo->frequenza)

- **Discretizzazione:** consiste nella conversione di un attributo continuo in uno ordinale  
Esistono due tipi di discretizzazione:
  - **Supervised:** vengono utilizzate le etichette di classe per individuare gli insiemi
  - **Unsupervised:** gli intervalli vengono definiti dopo l'analisi della distribuzione dei dati
    - N intervalli della stessa **dimensione**  $W = \frac{v_{\max} - v_{\min}}{N}$   
Questa tecnica è **facile** da implementare ed ha un approccio **incrementale** in quanto in presenza di nuovi valori mi basta aggiungere un bucket e non devo rivedere l'intera divisione.  
È limitato in presenza di **outlier e dati sparsi**.
    - N intervalli con approssimativamente la stessa **cardinalità**.  
Questa tecnica non ha un approccio incrementale ma gestisce meglio la presenza di outlier e dati sparsi.
    - **Clustering:** i gruppi sono identificati sulla base della distanza, funziona bene con dati sparsi e outliers
    - **Analisi della distribuzione:** divisione sulla base di quartili/percentili o altre divisioni di questo tipo

- **Binarizzazione:** un attributo viene mappato in una o più variabili binarie.

- Per attributi **continui**: deve prima essere mappato in un attributo categorico (altezza-> {basso, medio, alto})
- Per attributi **categorici**: viene effettuato il mapping ad un insieme di attributi binari. Ad esempio {basso, medio, alto} -> {100, 010, 001}, questo tipo di codifica in cui solo un bit per volta ha valore 1 è chiamato **one-hot encoding**.

- **Trasformazione di un attributo:** consiste in una funzione che mappa l'intero set di valori di un attributo ad un nuovo set, fanno parte di questa categoria:

- **Normalizzazione:** tecnica che **corregge la distanza** tra gli attributi in termini di frequenza di occorrenza, media, varianza, range. In alcuni casi consiste nel rappresentare i valori di un attributo in un range predefinito ([0, +1] o [-1, +1]). Le tecniche di normalizzazione più comuni sono:

- **Normalizzazione min-max:**

$$v' = \frac{v - min_A}{max_A - min_A} (new\_max_A - new\_min_A) + new\_min_A$$

Normalizza  $v'$  nell'intervallo  $[new\_min_A, new\_max_A]$ .

- **Normalizzazione z-score:**

$$v' = \frac{v - mean_A}{stand\_dev_A}$$

Mappa i valori tra  $[-\infty, +\infty]$ .

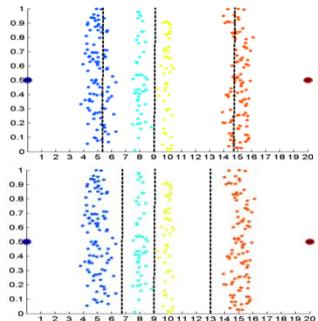
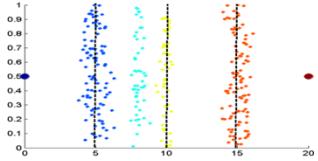
- **Scaling decimale:**

$$v' = \frac{v}{10^j}$$

dove j è l'intero più piccolo per cui  $\max(|v'|) < 1$

Mappa i valori tra  $[-1, +1]$ .

- **Standardizzazione:** consiste nel sottrarre il valore medio e dividere per la deviazione standard  $Z = \frac{x - \mu}{\sigma}$ , come viene fatto nello z-score



L'ultima figura rappresenta il clustering in 4 sottogruppi con attributi comuni (K-means)

## Similarità e dissimilarità

La **similarità** e la **dissimilarità** ci consentono di calcolare quanto simili o dissimili sono due oggetti, e vengono calcolate in diversi modi in base alle caratteristiche dell'attributo.

Esistono diverse tecniche per misurare la similarità fra due oggetti:

- **Distanza Euclidea:**  $d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}$  con  $n$  numero di attributi e  $x_k, y_k$  rispettivamente il  $k^{esimo}$  attributo degli oggetti  $x$  e  $y$ .

In questo caso la standardizzazione è necessaria se le scala è differente.

- **Distanza di Minkowski:**  $d(x, y) = (\sum_{k=1}^n |x_k - y_k|^r)^{\frac{1}{r}}$  in cui  $n$  è il numero di attributi e  $x_k, y_k$  rispettivamente il  $k^{esimo}$  attributo degli oggetti  $x$  e  $y$ .

$r$  è un parametro che può assumere valori tra 1 e inf.

- $r = 1$ : somma delle differenze degli attributi
- $r = \infty$ : distanza "suprema", indica la massima differenza fra gli attributi

### Proprietà delle distanze

Le proprietà comuni della **distanza** sono:

- **Positiva:**  $d(x, y) \geq 0$ , è uguale a zero solo se  $x = y$
- **Simmetria:**  $d(x, y) = d(y, x)$
- **Disugualanza triangolare:**  $d(x, z) \leq d(x, y) + d(y, z)$

Una formula della distanza che soddisfa queste proprietà è chiamata **metrica**.

### Proprietà della similarità

Le proprietà comuni della **similarità** sono:

- **Similarità massima:**  $s(x, y) = 1$  solo se  $x = y$
- **Simmetria:**  $s(x, y) = s(y, x)$

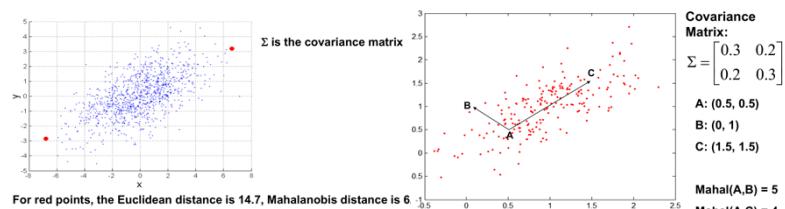
### Distanza di Mahalanobis

La distanza di Mahalanobis:

$$d(x, y) = (x - y)^T (\Sigma(x - y))^{-1}$$

con  $\Sigma$  che rappresenta la matrice di covarianza.

Consente di calcolare la distanza fra due punti rapportandola alla distribuzione di tutti gli altri punti.



### Similarità tra vettori binari

Le tecniche per misurare la similarità fra vettori **binari** sono:

dati due vettori binari  $p$  e  $q$

( $M_{ij}$  rappresenta il numero di attributi che in  $p$  assumono il valore  $i$  (0 o 1) e in  $q$  assumono il valore  $j$  (0 o 1))

- **Simple Matching:**  $SMC = \frac{M_{11} + M_{00}}{M_{01} + M_{10} + M_{11} + M_{00}} = \frac{\text{numero di similarità}}{\text{numero di attributi}}$
- **Jaccard Coefficients:**  $J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} = \frac{\text{numero di match che non hanno 00}}{\text{numero di match 11}}$

$$p = 10000000000$$

$$q = 0000001001$$

$$M_{01} = 2 \quad (\text{the number of attributes where } p \text{ was 0 and } q \text{ was 1})$$

$$M_{10} = 1 \quad (\text{the number of attributes where } p \text{ was 1 and } q \text{ was 0})$$

$$M_{00} = 7 \quad (\text{the number of attributes where } p \text{ was 0 and } q \text{ was 0})$$

$$M_{11} = 0 \quad (\text{the number of attributes where } p \text{ was 1 and } q \text{ was 1})$$

$$SMC = \frac{(M_{11} + M_{00})}{(M_{01} + M_{10} + M_{11} + M_{00})} = \frac{(0+7)}{(2+1+0+7)} = 0.7$$

$$J = \frac{M_{11}}{(M_{01} + M_{10} + M_{11})} = 0 / (2 + 1 + 0) = 0$$

Può essere utile per valutare la similarità relativa alla presenza della caratteristica  
**Cosine similarity**

La **cosine similarity** è utilizzata solitamente nei documenti testuali, quando sono rappresentati come vettori di parole, si calcola come  $\cos(d_1, d_2) = (d_1 \bullet d_2) / \|d_1\| \|d_2\|$  in cui  $\bullet$  indica il prodotto fra vettori e  $\|\cdot\|$  la loro norma.

### Combinazione delle similarità

Nella maggior parte dei casi gli **attributi** di un oggetto sono di tipo diverso, è quindi necessario **combinare le similarità**, lo si può fare in due modi:

- Definiamo una metrica di distanza definita come la **combinazione delle metriche** mediante l'indicatore  $\delta$  (che assume valori 0 se uno dei due oggetti ha un valore mancante rispettivamente all'attributo considerato o 1)
- Definiamo anche un peso  $\omega$  riguardante l'importanza che vogliamo dare ad ogni **singolo attributo**

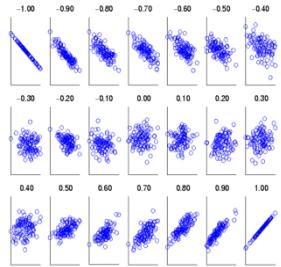
$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^n \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^n \delta_k}$$

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{k=1}^n \omega_k \delta_k s_k(\mathbf{x}, \mathbf{y})}{\sum_{k=1}^n \omega_k \delta_k}$$

### Correlazione di Pearson

L'analisi di **correlazione** permette di identificare se due variabili sono legate da una relazione lineare, quadratica, ecc utilizzando la matrice di correlazione.

Se c'è un'alta correlazione fra due attributi (solitamente  $\geq 0,8$ ) se ne considera solo uno, dato che l'altro è già rappresentato dal primo. Bisogna analizzare correttamente il segno della correlazione.



## 9. Regole di associazione

L'obiettivo dell'utilizzo delle regole di associazione è quello di estrarre correlazioni frequenti o specifici pattern all'interno di un database transazionale.

Dato un insieme di transazioni formato da diversi oggetti non ordinati, consideriamo una **regola di associazione**:

$$A, B \Rightarrow C$$

in cui A e B fanno parte del corpo della regola e C della testa e la freccetta indica la co-occorrenza.

L'estrazione delle regole di associazione è una tecnica **esplorativa** che ci permette di comprendere qualcosa in più sui dati ed è applicabile a qualsiasi tipo di dato. Particolare attenzione viene posta per i dati di tipo:

- **Dati testuali:** il documento viene considerato come una transazione e le parole ne sono gli oggetti
- **Dati strutturati:** una riga viene considerato come una transazione e le coppie attributo-valore ne sono gli oggetti (in caso di attributi continui, solitamente vengono discretizzati)

### Estrazione delle regole di associazione

Si definisce **itemset** un insieme di elementi.

Un itemset è **frequente** se ha un supporto maggiore di una determinata soglia.

### Metriche di qualità

Data una regola di associazione  $A \Rightarrow B$  con A e B definiti itemset definiamo due metriche di qualità:

- **Supporto:** percentuale di transazioni contenenti sia A che B

- **Confidenza:** percentuale di transazioni che presentano anche B rispetto alle transazioni che contengono solo A  $\frac{\text{sup}(A,B)}{\text{sup}(A)}$ , rappresenta la “forza” del “=>”

Dato un set di transazioni, l’association rule mining consiste nell’ estrarre regole di associazione che rispettano entrambe le condizioni:

- Supporto  $\geq \text{minsup}$  threshold
- Confidenza  $\geq \text{minconf}$  threshold

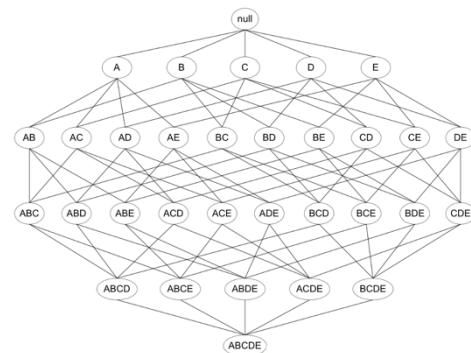
### Estrazione delle regole

L’estrazione delle regole di associazione viene eseguita in 2 fasi:

- **Estrazione degli itemset frequenti** che può essere fatta:
  - **A livelli:** (Apriori) vengono estratti prima gli itemset formati da un solo elemento, poi di due e così via
  - **Senza generazione di candidati:** (FP-growth) algoritmo ricorsivo che estrae gli itemset di interesse
  - Altri approcci
- **Estrazione delle regole di associazione** che consiste nella generazione di tutte le possibili combinazioni binarie di ogni itemset frequente utilizzando un threshold per la confidenza.

### Generazione degli itemset frequenti

Dati N oggetti, esistono  $2^N$  possibili itemset candidati che sono tutti rappresentati nel lattice.



### Brute-force

Il metodo **brute-force** è l’approccio base per la generazione degli itemset frequenti, consiste nell’esaminare tutto il database e calcolare il supporto di tutte le possibili combinazioni di itemset (l’intero lattice è considerato come candidato). Non è utilizzabile perché è computazionalmente troppo **oneroso**.

Per migliorarlo si può operare riducendo:

- Il numero di **candidati**: ridurre lo spazio di ricerca
- Il numero di **transazioni**: rimuovere le transazioni all’aumentare della dimensione dell’itemset
- Il numero di **confronti**: usare strutture dati più efficienti per memorizzare i candidati e le transazioni

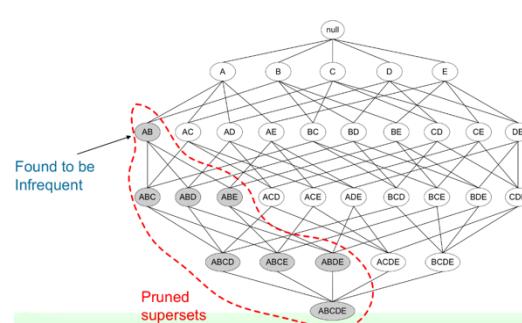
### Apriori

Il **principio di Apriori** dice che se un itemset è frequente, allora tutti i suoi subset sono frequenti. Allo stesso modo se un itemset è infrequente, anche tutti i superset lo saranno, per questo motivo non devono essere più considerati.

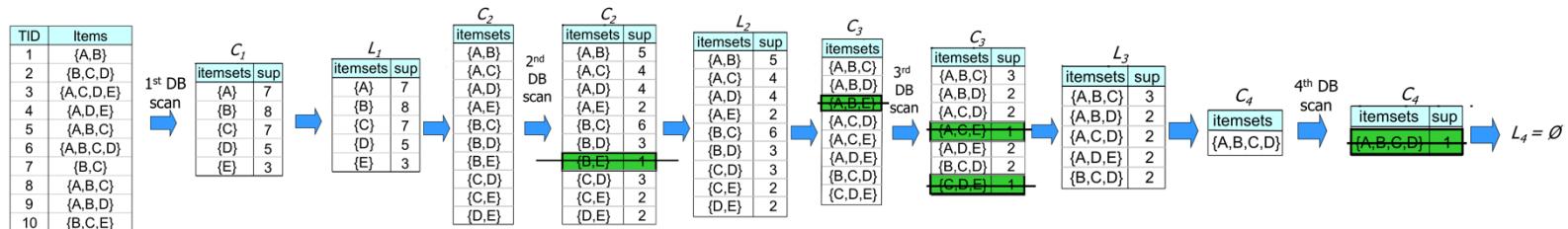
Questo principio viene impiegato all’interno dell’**algoritmo di Apriori** (Agr94) mediante un approccio a **livelli**.

Ad ogni iterazione vengono estratti gli itemset di lunghezza  $k$  e per ogni livello vengono eseguite due fasi:

- **Generazione dei candidati:**



- **Join**: vengono generati candidati di lunghezza  $k + 1$  unendo gli itemset frequenti di lunghezza  $k$  che hanno la parte iniziale lunga  $k - 1$  in comune
- **Prune**: applichiamo il principio di Apriori, eliminiamo i candidati di lunghezza  $k + 1$  che contengono almeno un  $k$ -itemset che non è frequente
- **Generazione degli itemset frequenti**: viene calcolato il supporto dei candidati  $k + 1$  e eliminiamo quelli sotto la soglia  $minsup$



Questo tipo di algoritmo può essere molto **efficiente** quando vogliamo individuare solo gli itemset frequenti di una certa lunghezza  **$k$  fissa**.

Le parti più critiche a livello computazionale per questo algoritmo possono essere:

- La generazione dei candidati
- E gli scan multipli del database per il conteggio del supporto

I fattori che comportano una limitazione delle performance sono:

- Il valore  $minsup$ 
  - Se **aumenta** troveremo solo correlazioni ovvie, ma non quelle più rare e interessanti
  - Se **diminuisce** troveremo troppe correlazioni e avremo un costo computazionale elevato
- Il numero di items del dataset
- Il numero di transazioni nel database
- La lunghezza media di ogni transazione

Per **limitare** il costo computazionale viene utilizzato un **hash-tree** che consente di limitare i costi di lettura della base dati.

Altri algoritmi che migliorano Agr94 sono:

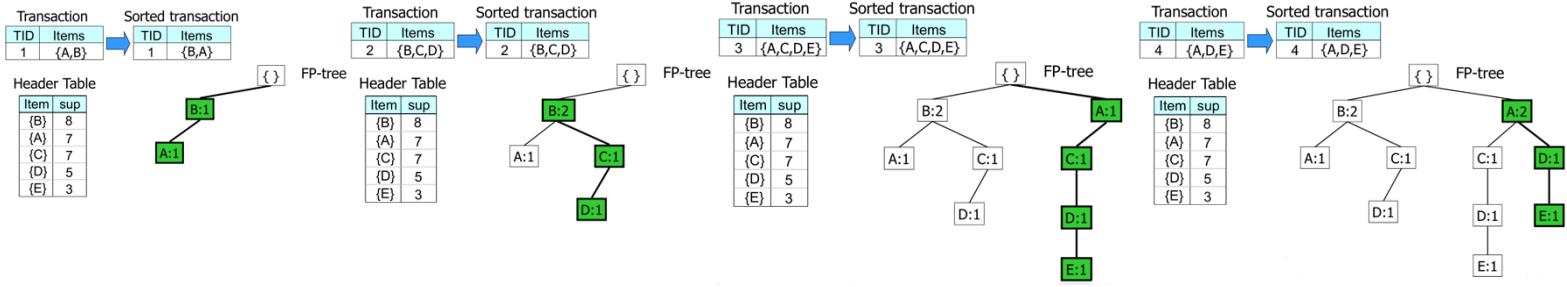
- **Yu95**:
  - Basa la selezione degli itemset frequenti sul conteggio degli itemset presenti nel bucket di hashing per aumentarne la velocità
  - Riduce l'impatto dello scan riducendo il numero di transazioni
- **Sav96**: consiste in una soluzione distribuita che lavora su un itemset partizionato
- **Toi96**: conteggio mediante sampling
- **Motw98**: conteggio dinamico

### FP-growth

L'algoritmo **FP-growth** (Han00) sfrutta una rappresentazione compatta della base di dati salvata su memoria in modo tale da non dover effettuare scansioni successive sul db. L'accesso al db è effettuato solo alla creazione della struttura dati e per il conteggio dei supporti.

La creazione del FP-tree avviene in tre fasi:

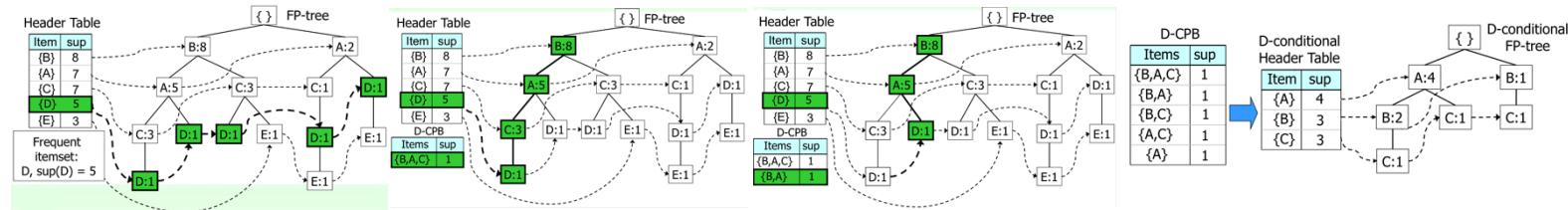
1. Conteggio dei **supporti** ed eliminazione degli oggetti al di sotto del *minsup*
2. Costruzione della **Header Table** ordinando gli oggetti in ordine di supporto decrescente
3. Creazione del **FP-tree**: per ogni transazione ordiniamo gli oggetti in ordine di supporto decrescente e li inseriamo all'interno dell'albero creando un nuovo branch quando la sequenza di oggetti cambia



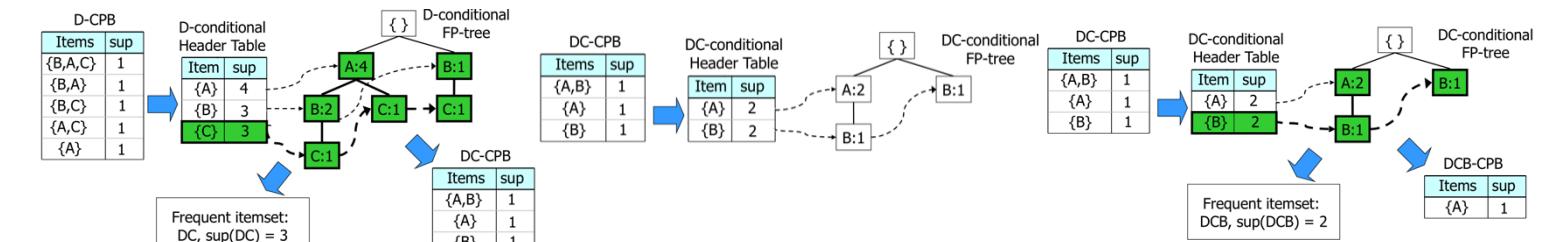
Nell'**FP-tree** finale avremo dei puntatori che “collegano” la Header Table con le foglie contenenti le istanze di un item e dei puntatori che collegano le foglie tra di loro. Questo meccanismo è chiamato **node link chain** ed è utile a velocizzare lo scanning.

L'algoritmo consiste nell'esplorare la Header Table in ordine di supporto a partire dal più basso, per ogni elemento *i* e i precedenti:

1. Costruiamo il **Conditional Pattern Base** per l'elemento *i* (i-CPB) selezionando i percorsi relativi ai prefissi di *i* dal FP-tree
2. Invochiamo ricorsivamente FP-growth su i-CPB

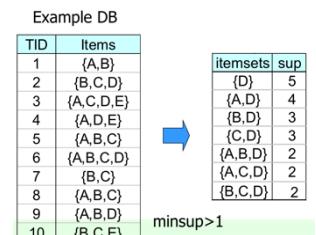


Seguendo l'esempio andiamo a considerare i prefissi relativi all'item D (avremo già fatto la stessa cosa per l'item E) per poi costruire un FP-tree specifico. Poi invochiamo ricorsivamente FP-growth su D-CPB, ottenendo gli alberi DC-CPB e in seguito DCB-CPB.



Notiamo che A non è frequente in DCB-CPB, quindi rimuoviamo A da DCB-CPB.

Ora torniamo indietro e generiamo l'albero per DCA-CPB, a poi torniamo su DB e DA analizzando ricorsivamente il loro interno. Alla fine, dopo le varie eliminazioni e dopo averle fatte per ogni item nella Header Table, otteniamo l'insieme di tutti gli itemset frequenti.

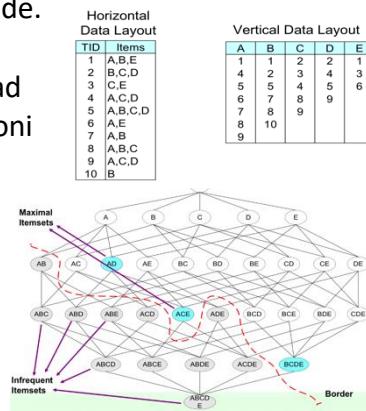


Questo approccio è efficiente quando la struttura in memoria non va a saturare la memoria stessa, inoltre in presenza di dati sparsi otteniamo un albero considerevolmente più grande.

Approcci differenti si basano su una diversa rappresentazione dei dati, ad esempio **inverted matrix** (Zak00) rappresenta per ogni item, la lista delle transazioni a cui appartiene.

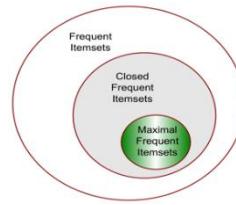
### Itemset frequente massimale e itemset closed

Un itemset è definito come frequente **massimale** se nessuno dei suoi superset direttamente collegati è frequente, guardando la figura non esiste nessun sovrainsieme di AD (ABD, ACD, ADE) frequente. Gli itemset frequenti massimali rappresentano gli itemset frequenti di lunghezza massima.



Un itemset si dice **closed** se nessuno dei suoi superset adiacenti ha lo stesso supporto dell'itemset.

Gli itemset massimali rappresentano un **sottoinsieme** dei closed. Essi sono utili per fare delle analisi esplorative quando gli item frequenti sono molto numerosi.



| TID | Items     | sup |
|-----|-----------|-----|
| 1   | (A,B)     | 4   |
| 2   | {B,C,D}   | 5   |
| 3   | (A,B,C,D) | 3   |
| 4   | (A,B,D)   | 4   |
| 5   | (A,B,C,D) | 2   |
|     | (A)       | 4   |
|     | (A,C)     | 3   |
|     | (A,D)     | 2   |
|     | (B,C)     | 3   |
|     | (B,D)     | 4   |
|     | (C,D)     | 3   |

| itemset   | sup |
|-----------|-----|
| (A,B,C)   | 2   |
| (A,B,D)   | 3   |
| (A,C,D)   | 2   |
| (B,C,D)   | 3   |
| (A,B,C,D) | 2   |

### Correlazione o lift

La misura della confidenza non è sempre affidabile (nei casi in cui la testa della relazione dovrebbe essere negativa per avere significato), infatti in alcuni casi potrebbe confondere, si noti l'esempio in figura.

Per far fronte a questo problema utilizziamo il concetto di **lift** (o correlazione), il quale viene associato ad ogni regola estratta.

**(Attenzione:** il lift (o correlazione) è diverso dalla correlazione di Pearson trattata nel capitolo precedente)

Viene calcolato come  $Lift = \frac{P(A,B)}{P(A)P(B)} = \frac{conf(r)}{sup(B)}$  con  $r: A \Rightarrow B$ ,

se questo valore si trova:

- All'interno di un certo intorno di 1 allora il corpo e la testa della regola sono considerati completamente incorrelati.
- Se il valore è molto minore di 1, allora abbiamo una correlazione negativa che verrà utilizzata mediante una regola inversa  $A \Rightarrow !B$ .
- Se il valore è maggiore di 1 la correlazione è positiva e la regola non deve essere modificata

In alcuni casi ogni transazione o oggetto può avere una diversa importanza, è opportuno quindi considerare, se necessario, un **peso** associato al singolo oggetto o transazione che può essere utilizzato anche per il calcolo di un supporto o di una confidenza pesata e necessita funzioni di aggregazioni pesate.

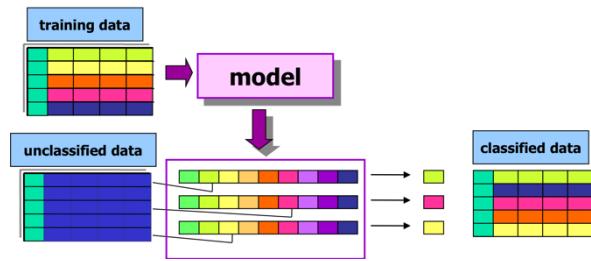
In alcuni casi potrebbe essere utile considerare, all'interno delle regole, una loro **generalizzazione** detta **tassonomia** utilizzando lo stesso concetto delle gerarchie. Possiamo quindi avere anche regole **cross** che contengono in A un oggetto e in B una generalizzazione. In questo caso l'estrazione degli itemset frequenti ci restituisce regole a diverse granularità.

5000 high school students are given  
 • 3750 eat cereals  
 • 3000 play basket  
 • 2000 eat cereals and play basket  
**Rule**  
 play basket  $\Rightarrow$  eat cereals  
 $sup = 40\%$ ,  $conf = 66.7\%$   
 is misleading because eat cereals has sup 75% ( $> 66.7\%$ )  
 ▪ Problem caused by high frequency of rule head  
 • negative correlation

|             | basket | not basket | total |
|-------------|--------|------------|-------|
| cereals     | 2000   | 1750       | 3750  |
| not cereals | 1000   | 250        | 1250  |
| total       | 3000   | 2000       | 5000  |

## 10. Classificazione

L'obiettivo della classificazione è quello di predire e assegnare un'**etichetta di classe** (presa da un insieme predefinito di etichette di classe) mediante un modello interpretabile generato sulla base di un dataset di training e validato mediante un dataset di test, per questo motivo è definito come metodo **supervised**.



Le diverse tecniche di classificazione vengono valutate secondo diversi fattori:

- **Accuratezza**: qualità delle predizioni
- **Interpretabilità**: interpretabilità e compattezza del modello
- **Incrementalità**: il modello si aggiorna dinamicamente in presenza di nuovi dati classificati aggiunti al dataset di training
- **Efficienza**: tempo di costruzione del modello e di classificazione
- **Scalabilità**: performance dell'algoritmo rispetto a dimensione del training set e numero di attributi
- **Robustezza**: capacità dell'algoritmo di operare correttamente anche in presenza di rumore e dati mancanti

### Albero decisionale

L'**albero decisionale** è una struttura che permette di determinare l'etichetta di classe di un elemento sfruttando di volta in volta delle condizionalità (presenti nei nodi) sui valori degli attributi, scorrendo un albero fino ad arrivare alle foglie che rappresentano le classi.

Dato un dataset è possibile avere **diversi** alberi decisionali che soddisfino la classificazione voluta.

Mentre nella fase di training avviene la creazione effettiva dell'albero, durante la fase di **test** si applica il modello ai dati di test per capire quanto esso sia accurato.

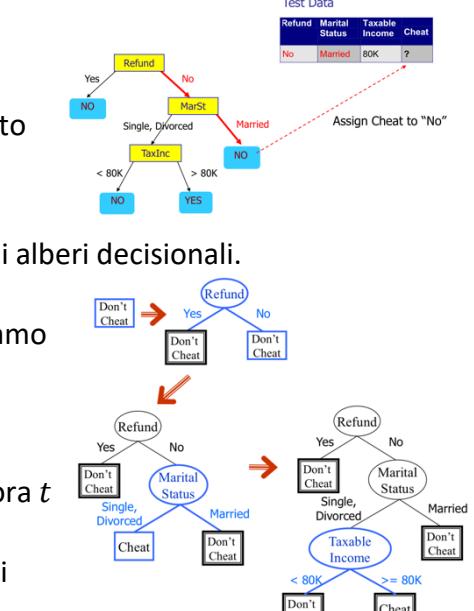
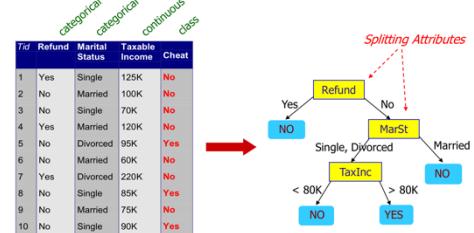
### Algoritmo di Hunt

L'**algoritmo di Hunt** è uno dei primi algoritmi utilizzati per la creazione di alberi decisionali.

Chiamando  $D_t$  l'insieme di record di training che raggiungono il nodo  $t$ :

- Se  $D_t$  contiene record appartenenti a **più di una classe** selezioniamo l'attributo A **migliore** per lo split e nominiamo il nodo  $t$  con A, in seguito dividiamo  $D_t$  in subset più piccoli e applichiamo **ricorsivamente** questa procedura
- Se  $D_t$  contiene record che appartengono alla **stessa classe**  $y_t$  allora  $t$  diventa un nodo foglia etichettato con  $y_t$ .
- Se  $D_t$  è vuoto allora  $t$  è un nodo foglia etichettato con la classe di maggioranza  $y_d$

Quando ho finito gli attributi da analizzare ma i record sono ancora eterogenei assegno la label che ha la maggioranza all'interno dell'insieme.



L'algoritmo di Hunt è un algoritmo interpretabile ma non incrementale, in quanto andrebbe rieseguito in presenza di nuovi record.

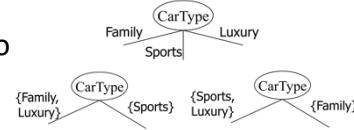
Presenta diverse tematiche da attenzionare:

- **Scelta della struttura della condizione di test:**

Dipende dal tipo di attributo:

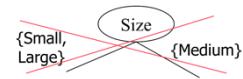
- Nominale:

- Split multiplo: creo un ramo per ogni possibile valore distinto
- Split binario: divido i valori in due subset e trovo il partizionamento ottimale



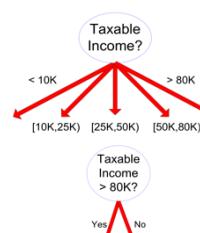
- Ordinale:

- Analogi ai nominali ma nello split binario ma nello stesso subset i valori devono essere continui



- Continuo:

- **Discretizzazione** utile a generare un attributo categorico ordinale  
La discretizzazione può essere fatta sia prima che durante la creazione dell'albero e può essere fatta utilizzando i metodi visti in precedenza.
- **Decisione binaria:**  $(A < \nu)$  o  $(A \geq \nu)$  bisogna individuare però lo split migliore.



- **Selezione dell'attributo migliore per lo split:**

Durante questa fase sono preferibili attributi con distribuzione delle classi **omogenea**, per far ciò serve misurare la “**purezza**” dell'attributo e possiamo farlo mediante diverse metriche:

|       |       |
|-------|-------|
| c0: 5 | c1: 5 |
|-------|-------|

Non-homogeneous, high degree of impurity

|       |       |
|-------|-------|
| c0: 9 | c1: 1 |
|-------|-------|

Homogeneous, low degree of impurity

- **GINI index** (valori da 0 a 1)

- **Entropy** (valori da 0 a  $\log_2 n_c$ ): la computazione è simile a quella del GINI.

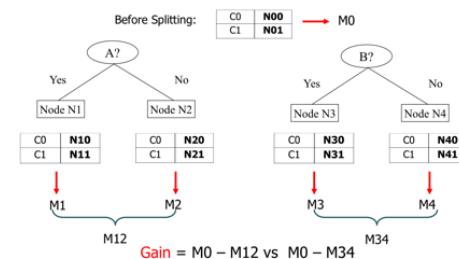
- **Information gain**: misura la riduzione dell'entropia data dallo split. Tende a dare maggiore importanza a split formati da molti gruppi piccoli e puri.

- **GainRatio**: aggiustamento dell'information gain per penalizzare gli split multipli e piccoli.

- **Misclassification error**

Verifichiamo sperimentalmente quale di queste metriche è la migliore per il nostro dataset.

L'esempio mostra come, scelta una metrica è possibile valutare quale split è il più **efficace**. Dopo aver calcolato l'impurità prima dello splitting e quella dopo lo splitting relativa ad entrambi gli attributi confrontiamo le differenze di impurità (gain) e scegliamo lo split con un gain maggiore.



Il **GINI index** serve per calcolare l'impurità di un nodo figlio si calcola come

$$GINI(t) = 1 - \sum_j [p(j|t)]^2$$

in cui  $p(j|t) = \frac{\text{elementi}_{\text{classe}j}}{\text{elementi}_{\text{totali}}}$  è la frequenza relativa della classe  $j$  rispetto al nodo  $t$ .

Più l'indice di GINI è piccolo, minore è il grado di impurità.

|                   |   |
|-------------------|---|
| C1                | 0 |
| C2                | 6 |
| <b>Gini=0.000</b> |   |

|                   |   |
|-------------------|---|
| C1                | 1 |
| C2                | 5 |
| <b>Gini=0.278</b> |   |

|                   |   |
|-------------------|---|
| C1                | 2 |
| C2                | 4 |
| <b>Gini=0.444</b> |   |

|                   |   |
|-------------------|---|
| C1                | 3 |
| C2                | 3 |
| <b>Gini=0.500</b> |   |

In uno split abbiamo **n nodi** possibili quindi utilizziamo

$$GINI_{split} = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

con  $n_i$  numero di record al figlio  $i$  e  $n$  numero di record al nodo  $p$ . Serve a calcolare un GINI index **pesato** rispetto alla numerosità dei record nei nodi figli.

- Attributi **booleani**: il calcolo avviene normalmente per i due nodi figli
- Attributi **categorici**: per ogni valore distinto contiamo i valori di ciascuna classe e decidiamo se effettuare uno split multiplo o binario in base alla matrice
- Attributi **continui**: se la discretizzazione non è avvenuta già precedentemente l'algoritmo valuta qual è la condizione migliore confrontando i GINI index relativi a tutti i possibili valori per tutte le possibili condizioni, scegliendo lo split con GINI index più piccolo.

The diagram illustrates the process of creating a decision tree. It starts with a 'Multi-way split' showing a matrix of 'CarType' values (Family, Sports, Luxury) for nodes C1 and C2. The GINI index for C1 is 0.393. A vertical dashed line separates this from a 'Two-way split' where the data is partitioned into three classes: (Sports, Family), (Sports, Luxury), and (Family, Luxury). The GINI index for the first node in the (Sports, Family) partition is 0.400. Below this, a detailed view of the 'Two-way split' for 'CarType' shows 'Sorted Values' and 'Split Positions' for 'Cheat' and 'Taxable Income'. The 'Yes' row has GINI values ranging from 0.375 to 0.420, while the 'No' row has values from 0.300 to 0.420.

- **Criteri di stop** nella creazione di foglie all'interno dell'albero, abbiamo diverse possibilità:
  - Fermiamo l'espansione di un nodo quando tutti i suoi record appartengono alla stessa classe
  - Fermiamo l'espansione di un nodo quanto tutti i record hanno valori di attributi simili
  - Conclusione anticipata:
    - Pre-pruning: durante la fase di creazione dell'albero noto che c'è un numero sufficientemente alto di nodi e mi fermo
    - Post-pruning: una volta creato l'albero completo decido quali sezioni o nodi togliere

Si nota che all'aumentare del numero di nodi il modello diventa sempre più accurato ma fino ad una certa soglia, superata la quale si parla di **overfitting**. Se invece i nodi sono troppo pochi parliamo di **underfitting**.

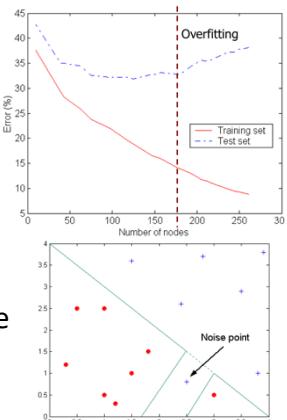
In alcuni casi, come quello mostrato in figura, l'overfitting dipende dalla presenza di rumore, in questi casi conviene avere un modello meno preciso che vada ad identificare semplicemente la diagonale.

La presenza di **valori mancanti** può influenzare la costruzione dell'albero decisionale nei seguenti modi:

- Influenza il calcolo dell'impurità
- Non si sa come indirizzare il record su un certo ramo se lo specifico attributo non è indicato
- Non si sa come classificare un'istanza di test con valori mancanti

**Valutazione degli alberi decisionali:**

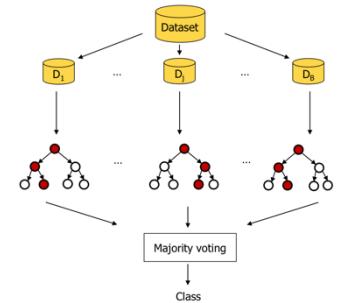
- **Accuratezza**: per dataset semplici, è comparabile ad altre tecniche di classificazione
- **Interpretabilità**: il modello è interpretabile quando l'albero non è molto grande e le singole predizioni sono interpretabili
- **Incrementalità**: non è incrementale
- **Efficienza**: la costruzione del modello e la classificazione sono veloci
- **Scalabilità**: è scalabile sia rispetto alla dimensione del training set che rispetto al numero di attributi



- **Robustezza:** la gestione dei dati mancanti è critica

## Random forest

Il **random forest** è un metodo che prevede la generazione di diversi alberi decisionali che lavorano insieme migliorando l'accuratezza generale e evitando l'overfitting. In fase di training vengono generati **diversi** alberi che classificheranno in maniera autonoma i record, la classe effettiva viene assegnata per **majority voting**.



Nello specifico il dataset viene diviso in subset **random** i cui record sono estratti con **rimpiattamento**, per il quale viene selezionato anche un subset di attributi random (solitamente se  $p$  è il numero di attributi vengono selezionati  $\sqrt{p}$  attributi) e viene generato per ognuno dei subset un albero. Gli alberi ottenuti sono completamente **decorrelati**. La classe viene assegnata per maggioranza fra tutte le predizioni dei diversi alberi.

### Valutazione del random forest:

- **Accuratezza:** maggiore rispetto agli alberi decisionali
- **Interpretabilità:** l'interpretabilità diminuisce all'aumentare del numero di alberi, ma permette di capire l'importanza delle features rispetto a tutte le altre
- **Incrementalità:** non è incrementale
- **Efficienza:** risulta più veloce rispetto agli alberi decisionali perché una volta selezionati i subset le operazioni di costruzione degli alberi e di classificazione saranno più veloci perché eseguibili contemporaneamente
- **Scalabilità:** è scalabile in quanto gli alberi vengono creati su una porzione di dati
- **Robustezza:** è robusto rispetto a rumore e outliers perché abbiamo modelli diversi che lavorano in contemporanea e lavora meglio in presenza di missing values

## Rule Based classification

La **Rule-based classification** è basata sul concetto di regole di classificazione.

Il modello di predizione dell'etichetta di classe è composto da un insieme di regole scritte nella forma (*Condition*)  $\rightarrow$  *y*, in cui la condizione è rappresentata da un insieme di predicati e *y* rappresenta l'etichetta di classe assegnata.

Esistono degli **algoritmi** che funzionano bene in presenza di dati strutturati che dato in input un dataset di training, estrapolano tutte le regole di classificazione.

| Name         | Blood Type | Give Birth | Can Fly | Live in Water | Class      |
|--------------|------------|------------|---------|---------------|------------|
| human        | warm       | yes        | no      | no            | mammals    |
| salmon       | cold       | no         | no      | no            | reptiles   |
| whale        | warm       | yes        | yes     | yes           | mammals    |
| frog         | cold       | no         | no      | sometimes     | amphibians |
| komodo       | cold       | no         | no      | no            | reptiles   |
| bat          | warm       | yes        | yes     | no            | mammals    |
| pigeon       | warm       | no         | yes     | no            | birds      |
| cat          | warm       | yes        | no      | no            | mammals    |
| turtle       | cold       | no         | no      | sometimes     | reptiles   |
| penguin      | warm       | no         | no      | sometimes     | birds      |
| porcupine    | warm       | yes        | no      | no            | mammals    |
| eel          | cold       | yes        | no      | yes           | fishes     |
| salamander   | cold       | no         | no      | sometimes     | amphibians |
| gila monster | cold       | no         | no      | no            | reptiles   |
| platypus     | warm       | no         | no      | no            | mammals    |
| owl          | warm       | no         | yes     | no            | birds      |
| dolphin      | warm       | yes        | yes     | yes           | mammals    |
| eagle        | warm       | no         | no      | no            | birds      |

R1: (Give Birth = no)  $\wedge$  (Can Fly = yes)  $\rightarrow$  Birds  
R2: (Give Birth = no)  $\wedge$  (Live in Water = yes)  $\rightarrow$  Fishes  
R3: (Give Birth = yes)  $\wedge$  (Blood Type = warm)  $\rightarrow$  Mammals  
R4: (Give Birth = no)  $\wedge$  (Can Fly = no)  $\rightarrow$  Reptiles  
R5: (Live in Water = sometimes)  $\rightarrow$  Amphibians

Una regola **copre** un record se gli attributi del record soddisfano le condizioni della regola.

Nella condizione **ideale** abbiamo regole:

- **Mutualmente esclusive:** due regole non possono essere vere contemporaneamente e ogni record deve essere coperto al massimo da una regola
- **Esaustive:** le regole di classificazione devono considerare tutte le possibili combinazioni di valori di attributi e ogni record deve essere coperto almeno da una regola

Ci sono dei meccanismi che permettono di estrarre le regole direttamente da un **albero decisionale**, ogni path dalla radice ad una foglia rappresenta una regola. In questo caso le condizione **ideale** sarà rispettata.

Nel caso in cui l'albero decisionale è il risultato di un'operazione di pruning le regole estratte avranno un contenuto informativo non ideale.

C'è anche la possibilità di semplificare le regole e lo si può fare in tre modi:

- Semplificando le regole estratte dall'albero completo
- Estraendo le regole da un albero decisionale prunato
- Estraendo le regole direttamente dal training set

Fare ciò comporta la perdita dei requisiti di mutua esclusività e di esaustività, in questo caso:

- Se un record soddisfa **più** di una regola bisogna trovare un meccanismo di scelta
  - Viene considerata la classe relativa al primo match
  - Viene presa la classe maggioritaria
- Se un record non soddisfa **nessuna** regola gli viene assegnata una classe di default

Ci sono due diversi metodi per l'individuazione delle regole di classificazione:

- Metodo **diretto**: le regole vengono estratte direttamente dai dati
- Metodo **indiretto**: le regole vengono estratte da modelli di classificazione pre-esistenti

Valutazione delle regole di classificazione:

- **Accuratezza**: maggiore rispetto agli alberi decisionali e al random forest quando le regole vengono estratte direttamente dal dataset e non da un modello pre-esistente
- **Interpretabilità**: il modello e le predizioni sono interpretabili
- **Incrementalità**: non è incrementale
- **Efficienza**: è veloce sia la costruzione del modello che la classificazione
- **Scalabilità**: è scalabile sia dal punto di vista della cardinalità del training set sia da quello del numero di attributi
- **Robustezza**: è robusto rispetto agli outliers

### *Classificazione associativa*

La **classificazione associativa** estende la classificazione di tipo rule-based sfruttando le regole di **associazione** per effettuare una predizione. Vengono però prese in considerazione solo le regole associative del tipo (*Condition*) → *y*.

Il formato delle regole generate è lo stesso del rule-based, ciò che cambia è come esse vengono estratte (estrazione di regole con itemset frequenti).

Durante la generazione del modello le regole devono essere **selezionate** e **ordinate** rispetto a supporto, confidenza, correlazione e thresholds.

In seguito, per evitare **overfitting**, vengono prese in considerazione solo le regole più in alto nell'ordinamento mentre tutte le altre vengono pruned, sussiste il problema relativo alla definizione dei thresholds relativi alle variabili.

Valutazione della classificazione associativa:

- **Accuratezza**: maggiore rispetto agli alberi decisionali, al random forest e al rule-based in quanto viene considerata la **correlazione** fra gli attributi
- **Interpretabilità**: il modello e le predizioni sono interpretabili
- **Incrementalità**: non è incrementale
- **Efficienza**: la creazione del modello è lenta per via dell'onerosità dell'estrazione degli itemset frequenti, la classificazione è invece molto veloce

- **Scalabilità:** è scalabile rispetto alla cardinalità del training set mentre lo è meno rispetto al numero di attributi, in quanto la generazione delle regole diventa più onerosa
- **Robustezza:** è robusto rispetto agli outliers e non è affatto da problemi relativi a dati mancanti

### *k-nearest neighbor*

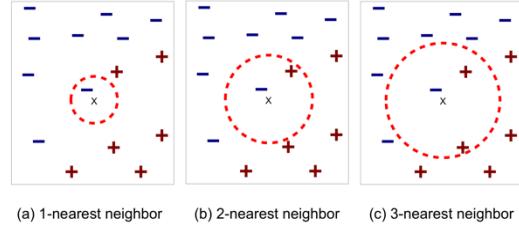
La tecnica del **k-nearest neighbor** utilizza il dataset di training per predire l'etichetta di classe senza la necessità di un modello predittivo. Quando un nuovo oggetto deve essere classificato vengono identificati i  $k$  oggetti del training set considerati vicini secondo una certa metrica di distanza e assegnamo la classe che identifica il maggior numero di vicini.

L'attuazione di questa tecnica richiede:

- Il training set
- Un metrica che permetta di calcolare la distanza fra i record
- Il valore  $k$ , numero di vicini da individuare

Per la classificazione dei record:

- Calcolare la distanza rispetto ai record di training
- Identificare i  $k$  vicini
- Assegnare la classe a partire dalle classi dei vicini



È possibile anche considerare una metrica pesata o pesare il voto rispetto alla distanza  $w = \frac{1}{d^2}$

La scelta del  $k$ , che deve essere fatta mediante tentativi successivi, è cruciale in quanto:

- Se è troppo **piccolo** abbiamo una alta sensitività al rumore
- Se è troppo **grande** è probabile che il vicinato includa anche punti appartenenti ad altre classi

Valutazione della **k-nearest neighbor**:

- **Accuratezza:** comparabile alle altre tecniche di classificazione per dataset semplici
- **Interpretabilità:** il modello non è interpretabile ma le singole predizioni possono essere descritte dai vicini
- **Incrementalità:** è incrementale
- **Efficienza:** la fase di training è inesistente dato che non viene generato un modello, mentre la classificazione è molto lenta perché richiede il calcolo delle distanze per ogni punto
- **Scalabilità:** non è scalabile dal punto di vista sia della numerosità del training set (perché calcola le distanze per ogni record) sia da quello della numerosità delle dimensioni ([curse of dimensionality](#)) che rende sempre più difficile il calcolo delle distanze, inoltre il dominio degli attributi deve essere normalizzato per evitare la predominanza di certi attributi sul calcolo della distanza
- **Robustezza:** dipende dal metodo utilizzato per il calcolo della distanza

### *Classificazione Bayesiana*

La **classificazione Bayesiana** si basa sul teorema di Bayes per l'individuazione dell'etichetta di classe.

Consideriamo tutti gli attributi come variabili casuali  $x_1, x_2, \dots, x_k$  con il record rappresentato da  $X = \langle x_1, x_2, \dots, x_k \rangle$ , mentre  $C$  rappresenta l'etichetta di classe.

La classificazione Bayesiana calcola  $P(C|X) = \frac{P(X|C)*P(C)}{P(x)}$  che rappresenta la probabilità di assegnare al record  $X$  la classe  $C$  calcolata sfruttando il Teorema di Bayes.  
La classe **assegnata** a  $X$  è la classe con per cui  $P(C|X)$  è il **massimo**.

$P(X)$  è costante  $\frac{1}{N}$  (perché non abbiamo record duplicati) e non lo consideriamo nel calcolo finale.  
 $P(C)$  viene calcolato come  $\frac{N_C}{N}$  con  $N_C$  numero di record appartenenti alla classe  $C$  e  $N$  numero di record totali.

Il calcolo di  $P(X|C)$  avviene applicando l'ipotesi Naive (di indipendenza statistica degli attributi)  
 $P(X|C) = P(x_1, \dots, x_k|C) = P(x_1|C) * P(x_2|C) \dots * P(x_k|C)$

In cui  $P(x_n|C)$  viene calcolato:

- Per funzioni **discrete**:  $P(x_n|C) = \frac{|x_{ncl}|}{N_C}$  con il numeratore che rappresente il numero di istanze aventi valore  $x_n$  per l'attributo  $n$  che appartengono alla classe  $C$
- Per funzioni **continue** invece usiamo la distribuzione di probabilità

| Outlook  | Temperature | Humidity | Windy | Class |
|----------|-------------|----------|-------|-------|
| sunny    | hot         | high     | false | N     |
| sunny    | hot         | high     | true  | N     |
| overcast | hot         | high     | false | P     |
| rain     | mild        | high     | false | P     |
| rain     | cool        | normal   | false | P     |
| rain     | cool        | normal   | true  | N     |
| overcast | cool        | normal   | true  | P     |
| sunny    | mild        | high     | false | N     |
| sunny    | cool        | normal   | false | P     |
| rain     | mild        | normal   | false | P     |
| sunny    | mild        | normal   | true  | P     |
| overcast | mild        | high     | true  | P     |
| overcast | hot         | normal   | false | P     |
| rain     | mild        | high     | true  | N     |

| outlook                      |                            |  |
|------------------------------|----------------------------|--|
| $P(\text{sunny} p) = 2/9$    | $P(\text{sunny} n) = 3/5$  |  |
| $P(\text{overcast} p) = 4/9$ | $P(\text{overcast} n) = 0$ |  |
| $P(\text{rain} p) = 3/9$     | $P(\text{rain} n) = 2/5$   |  |
| temperature                  |                            |  |
| $P(\text{hot} p) = 2/9$      | $P(\text{hot} n) = 2/5$    |  |
| $P(\text{mild} p) = 4/9$     | $P(\text{mild} n) = 2/5$   |  |
| $P(\text{cool} p) = 3/9$     | $P(\text{cool} n) = 1/5$   |  |
| humidity                     |                            |  |
| $P(\text{high} p) = 3/9$     | $P(\text{high} n) = 4/5$   |  |
| $P(\text{normal} p) = 6/9$   | $P(\text{normal} n) = 2/5$ |  |
| windy                        |                            |  |
| $P(\text{true} p) = 3/9$     | $P(\text{true} n) = 3/5$   |  |
| $P(\text{false} p) = 6/9$    | $P(\text{false} n) = 2/5$  |  |

$$\begin{aligned} P(p) &= 9/14 \\ P(n) &= 5/14 \end{aligned}$$

Data to be labeled

$X = \langle \text{rain}, \text{hot}, \text{high}, \text{false} \rangle$

For class p

$$\begin{aligned} P(X|p) \cdot P(p) &= \\ &= P(\text{rain}|p) \cdot P(\text{hot}|p) \cdot P(\text{high}|p) \cdot P(\text{false}|p) \cdot P(p) \\ &= 3/9 \cdot 2/9 \cdot 3/9 \cdot 6/9 \cdot 9/14 = 0.010582 \end{aligned}$$

For class n

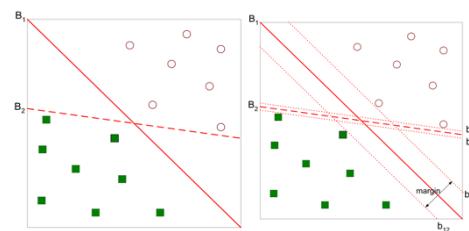
$$\begin{aligned} P(X|n) \cdot P(n) &= \\ &= P(\text{rain}|n) \cdot P(\text{hot}|n) \cdot P(\text{high}|n) \cdot P(\text{false}|n) \cdot P(n) \\ &= 2/5 \cdot 2/5 \cdot 4/5 \cdot 2/5 \cdot 5/14 = 0.018286 \end{aligned}$$

Valutazione della classificazione Bayesiana:

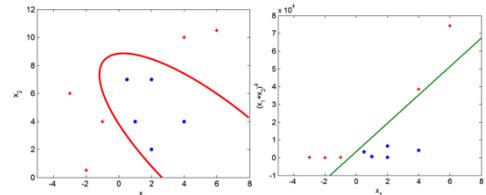
- **Accuratezza**: comparabile o più bassa degli alberi decisionali perché l'ipotesi Naive semplifica il modello
- **Interpretabilità**: il modello e la predizione non sono interpretabili, i pesi dei contributi in una singola predizione possono essere usati per la spiegazione
- **Incrementalità**: è incrementale e non richiede l'accesso al training set
- **Efficienza**: la costruzione del modello e la classificazione sono veloci
- **Scalabilità**: è scalabile sia dal punto di vista della dimensione del training set che da quello del numero di attributi
- **Robustezza**: dipende dalla correlazione effettiva fra gli attributi e da quanto l'ipotesi Naive sia effettivamente rispettata

### Support Vector Machine

Il metodo **Support Vector Machines** identifica un superpiano che separa le classi di interesse per creare un modello che distingua i record che appartengono a classi diverse. Ci sono diversi possibili iperpiani, la soluzione migliore viene identificata in quello che ha la dimensione dei margini maggiore.



Nel caso in cui i gruppi non siano suddivisi in maniera lineare vengono messi a disposizione dei meccanismi che rappresentano i dati in uno spazio diverso, nel quale è possibile identificare un modello lineare.



Valutazione del metodo **Support Vector Machines**:

- **Accuratezza:** molto alta
- **Interpretabilità:** il modello e la predizione non sono interpretabili
- **Incrementalità:** non è incrementale
- **Efficienza:** la costruzione del modello richiede tuning dei parametri, la classificazione invece è molto veloce
- **Scalabilità:** è mediamente scalabile sia dal punto di vista della dimensione del training set che da quello del numero di attributi
- **Robustezza:** è robusto rispetto a rumore e outliers

### *Artificial Neural Network*

Le **Artificial Neural Network** sono modelli ispirati alla struttura del cervello umano in cui i neuroni rappresentano le unità di elaborazione e le sinapsi rappresentano la rete di connessioni.

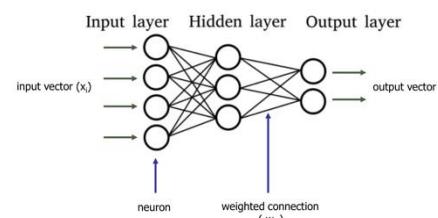
Esistono diverse tipologie di reti neurali:

- **Feed Forward Neural Network**

È formata da diversi livelli di neuroni in cui l'output di un livello rappresenta l'input del livello successivo che eseguirà a sua volta delle operazioni restituendo un output. Il dataset di training viene introdotto all'interno della rete neurale mediante il **livello di input**, mentre il **livello di output** restituisce l'etichetta di classe, all'interno del modello è possibile avere diversi **livelli di elaborazione**.

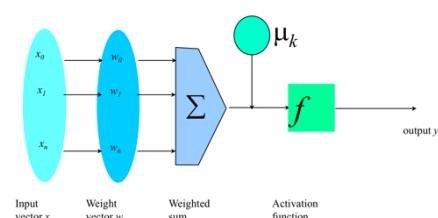
Questo modello è detto **fully connected** in quanto l'output di ogni neurone del livello precedente, viene dato in input a tutti i neuroni del livello successivo.

Alle connessioni fra i diversi nodi viene associato un **vettore peso**.



L'elaborazione effettuata da ogni neurone è la seguente:

- Riceve in input un vettore di valori e uno di pesi
- Effettua una **sintesi** degli input e dei pesi mediante una somma pesata di essi e la combina col suo offset
- Applica una **funzione di attivazione** che genera un output.

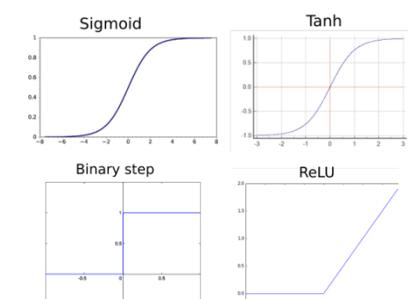


L'output dell'intera rete viene utilizzato per il calcolo dell'errore e la propagazione dello stesso avviene all'indietro mediante l'aggiornamento dei pesi nei vari livelli.

La **funzione di attivazione** simula l'attivazione biologica dei neuroni ad uno stimolo in input e fornisce non linearità alla computazione.

Esistono diverse funzioni di attivazione:

- **Sigmoid, tanh:** eseguono lo scaling dell'input fornendo un output in un determinato range ( $[0,1]$ ,  $[-1,1]$ )
- **Binary Step:** se il valore in input è negativo restituisce 0, se no restituisce 1



- **ReLU** (Rectified Linear Unit): se il valore in input è negativo restituisce 0, se è positivo restituisce un valore proporzionale. La particolarità di questa funzione è che non effettua uno scaling
- **Softmax**: è utilizzata solo per il livello di **output** e prende in considerazione tutti i neuroni del livello

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{i=0}^{N-1} e^{z_i}}$$

L'algoritmo di base per la costruzione di una **FFNN** è il seguente:

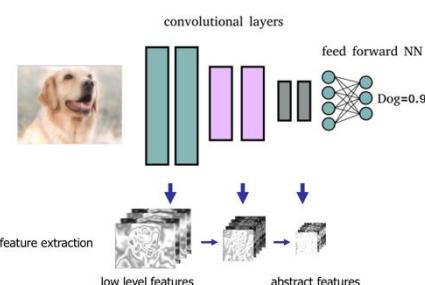
- Inizialmente vengono assegnati valori random per i pesi e gli offset
- Le istanze nel training set vengono processate una alla volta:
  - Per ogni neurone, viene calcolato il risultato applicando i pesi, gli offset e la funzione di attivazione per l'istanza
  - Viene propagato il risultato in avanti fino a che non viene calcolato l'output
  - A seguito della comparazione fra l'output calcolato e quello atteso, viene calcolato l'errore
  - L'errore viene propagato all'indietro aggiornando i pesi e l'offset per ogni neurone
- Il processo finisce in tre casi:
  - L'accuratezza supera un certo threshold
  - L'errore è minore di un certo threshold
  - Viene raggiunto il numero massimo prestabilito di epoch

Valutazione del metodo **FFNN**:

- **Accuratezza**: molto alta
- **Interpretabilità**: il modello e la predizione non sono interpretabili
- **Incrementalità**: non è incrementale
- **Efficienza**: la costruzione del modello richiede molto tempo e un tuning dei parametri molto complesso, la classificazione invece è molto veloce
- **Scalabilità**: è mediamente scalabile sia dal punto di vista della dimensione del training set che da quello del numero di attributi
- **Robustezza**: è robusto rispetto a rumore e outliers, richiede un training set molto grande

## • Convolutional Neural Network

Introduce dei **convolutional layer** che consentono di modellare i dati di input mediante **feature extraction**, l'output viene poi dato in pasto alla FFNN.

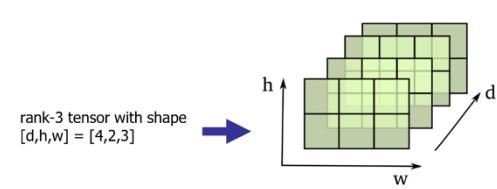


I vari livelli hanno la capacità di estrarre features con caratteristiche diverse:

- I **primi** livelli estraggono features molto specifiche del dato
- I livelli **successivi** estraggono features che tendono a sintetizzare il dato

I dati che passano attraverso una CNN si basano sul concetto di **tensore** che rappresenta un vettore multidimensionale

- **Rank**: rappresenta il numero di dimensioni:
  - **Scalare**: rank 0
  - Vettore 1-D: rank 1
  - Vettore 2-D: rank 2
- **Shape**: numero di elementi per ogni dimensione ( $h, w$ )



Un'immagine può essere rappresentata come un tensore con tre matrici per i colori RGB e una per la scala di grigi.

L'elaborazione dei singoli livelli convoluzionali segue tre fasi:

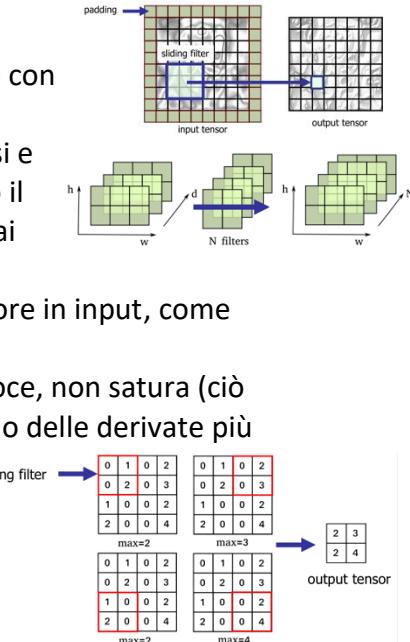
- **Convoluzione:** ottiene in input i dati di input iniziali oppure il tensore prodotto dal livello precedente e restituisce un tensore con le features estratte.

Questa operazione viene fatta mediante n filtri scorrevoli di pesi e produce un pixel (nel caso delle immagini) ottenuto elaborando il pixel precedentemente in quella posizione correlandolo anche ai pixel adiacenti.

- **Attivazione:** viene applicata una funzione di attivazione al tensore in input, come avviene nelle FFNN.

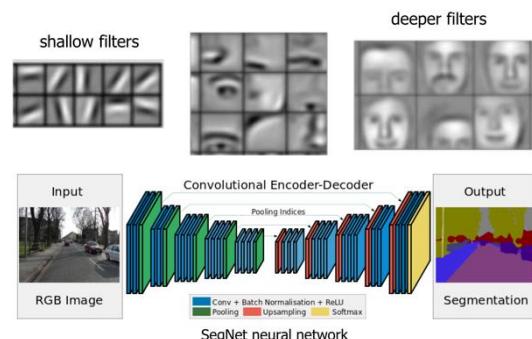
Tipicamente viene usata **ReLU** che consente un training più veloce, non satura (ciò consente la non propagazione degli errori) e permette un calcolo delle derivate più veloce per la propagazione all'indietro.

- **Pooling:** effettua un downsampling del tensore mediante un filtro scorrevole che rimpiazza i valori del tensore con una sintesi statistica degli output vicini. Viene solitamente utilizzato **maxpool** che considera il valore massimo.



Durante la fase di training ogni filtro impara a riconoscere specifici **pattern** nel tensore di input:

- Nei layer **superficiali** vengono riconosciuti la struttura e i bordi dell'immagine
- Nei layer più **profondi** vengono caratterizzati i singoli oggetti e le diverse parti dell'oggetto

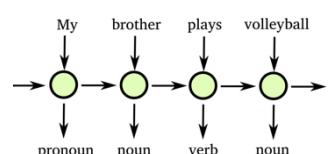


In pratica la CNN effettua una segmentazione semantica dell'immagine che rende la classificazione mediante FFNN più semplice e diretta.

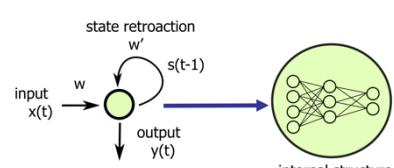
#### • Recurrent Neural Network

Consente l'elaborazione di dati che considerano il concetto di **sequenza**, ha la capacità di mantenere uno stato che evolve nel tempo. È utilizzato per diversi scopi:

- Traduzioni
- Predizione di serie temporali
- Trascrizione vocale
- Analisi grammaticale

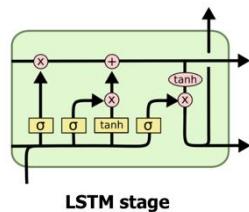


La rete riceve come input un vettore  $x(t)$  a lo stato allo step precedente  $s(t - 1)$ , il training viene fatto mediante **Backpropagation Through Time**: dato  $x(t)$  e l'output atteso  $y(t)$  l'errore viene propagato all'indietro nel tempo e i pesi vengono aggiornati in modo tale da minimizzare l'errore ad ogni step temporale.



RNN presenta il problema del **vanishing gradient** il quale decresce velocemente facendo in modo che i pesi non vengano aggiornati propriamente, rendendo difficile utilizzare memorie a lungo termine.

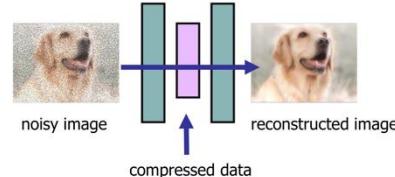
La soluzione a questo problema è **LSTM** (Long Short Term Memories) che utilizza delle porte logiche che permettono all'informazione sullo stato di attraversare lunghi intervalli di tempo.



- **Autoencoder**

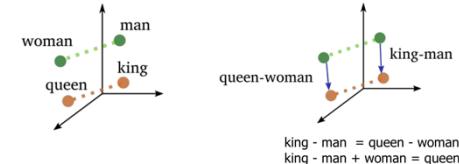
Consente la **compressione** dei dati in input e la successiva **ricostruzione** di essi, ha diversi utilizzi:

- **Feature extraction:** la rappresentazione compressa può essere usata come un set di features significative per la rappresentazione dei dati di input
- **Denoising** (per immagini e segnali)



- **Word Embeddings (Word2Vec)**

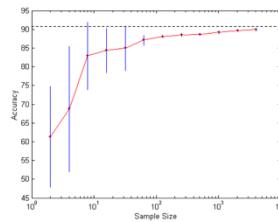
Le parole vengono rappresentate da vettori numerici che catturano l'informazione **semantica** e fanno in modo che parole con significati simili abbiano caratteristiche simili.



La vicinanza/lontananza semantica fra due parole viene calcolata mediante le posizioni dei vettori.

La **valutazione delle performance** di un modello può dipendere da diversi fattori oltre che dall'algoritmo scelto:

- Distribuzione dei record nelle classi
- Impatto sul modello di una classificazione errata
- Dimensione del training e del test set

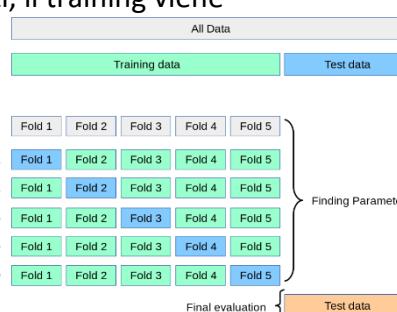


La **curva di apprendimento** mostra come l'accuracy cambia al variare della dimensione del training sample e mostra come un sample piccolo comporta una bassa accuratezza e un'alta deviazione dell'errore.

## Creazione del training set e test set

Esistono diversi modi per il partizionamento dei dati in training e test set:

- **Sampling con o senza rimpiazzamento**
- **Holdout:** consiste in un partizionamento fisso (tipicamente 80-20), è più appropriato per db di grandi dimensioni
- **Cross validation (k-fold):** i dati vengono partizionati in  $k$  subset disgiunti, il training viene eseguito su  $k - 1$  subset e il test sul subset rimanente e l'operazione viene ripetuta per tutte le  $k$  combinazioni possibili. Fornisce una stima dell'accuracy affidabile ma non è appropriato per dataset grandi



Un caso particolare di cross validation ( $k = n$ ) è il **leave-one-out** in cui il test set è composto da un solo elemento, è appropriato solo per database molto piccoli.

## Validazione dei modelli

Per stimare l'**accuratezza** di un algoritmo bisogna introdurre una fase di **validazione** fra la fase di training e quella di test dell'algoritmo. In questa fase, a cui solitamente viene riservata una parte del training set, viene fatta un'analisi di **sensitività** ai parametri propri dell'algoritmo e una comparativa fra i risultati dei diversi algoritmi.

Le diverse metriche per la valutazione delle performance sfruttano la **matrice di confusione** che rappresenta il numero di predizioni effettuate rispetto alle classi effettive.

|              |           | PREDICTED CLASS |           |
|--------------|-----------|-----------------|-----------|
|              |           | Class=Yes       | Class=No  |
| ACTUAL CLASS | Class=Yes | a<br>(TP)       | b<br>(FN) |
|              | Class=No  | c<br>(FP)       | d<br>(TN) |

a: TP (true positive)  
b: FN (false negative)  
c: FP (false positive)  
d: TN (true negative)

La metrica più usata è quella dell'**accuratezza**:  $accuracy = \frac{\text{Number of correctly classified objects}}{\text{Number of classified objects}}$

Essa risulta inaffidabile quando abbiamo dei dati con classi particolarmente sbilanciate.

In un dataset contenente il 99% dei dati con classe 0 e l'1% dei dati con classe 1, un modello che classifica tutti i record come appartenenti alla classe 0 ha un'accuratezza del 99%.

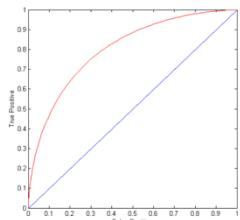
Possibili metriche per una valutazione all'interno della determinata classe sono:

- $Recall(r) = \frac{\text{Number of objects correctly assigned to } C}{\text{Number of objects belonging to } C} = \frac{TP}{TP+FN}$  che rappresenta la capacità del modello di riconoscere gli oggetti che effettivamente appartengono alla classe.
- $Precision(p) = \frac{\text{Number of objects correctly assigned to } C}{\text{Number of objects assigned to } C} = \frac{TP}{TP+FP}$  che rappresenta la capacità del modello di essere preciso nel fare la propria predizione rispetto alla classe di interesse.
- $F1_{score} = 2 * \frac{r*p}{r+p}$

### Curva ROC

Una tecnica per la comparazione dei modelli è la curva **ROC** (Receiver Operating Characteristic) che plotta in un grafico:

- TPR, True Positive Rate  $TPR = \frac{TP}{TP+FN}$
- FPR, False Positive Rate  $FPR = \frac{FP}{FP+TN}$

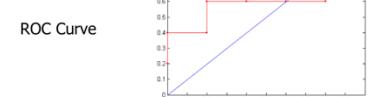


In cui il punto (0,1) rappresenta la situazione ideale e la diagonale rappresenta un modello che predice l'etichetta di classe randomicamente. Il modello con l'area maggiore è quello che performa meglio.

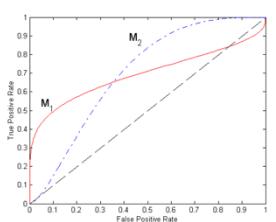
| Class    | +    | -    | +    | -    | -    | -    | +    | -    | +    | +    | 0.00 |
|----------|------|------|------|------|------|------|------|------|------|------|------|
| $P(+ A)$ | 0.25 | 0.43 | 0.53 | 0.76 | 0.89 | 0.85 | 0.85 | 0.87 | 0.93 | 0.95 | 1.00 |
| TP       | 5    | 4    | 4    | 3    | 3    | 3    | 3    | 2    | 1    | 1    | 0    |
| FP       | 5    | 5    | 4    | 4    | 3    | 2    | 1    | 1    | 0    | 0    | 0    |
| TN       | 0    | 0    | 1    | 1    | 2    | 3    | 4    | 4    | 5    | 5    | 5    |
| FN       | 0    | 1    | 1    | 2    | 2    | 2    | 2    | 3    | 3    | 4    | 5    |
| TPR      | 1    | 0.8  | 0.8  | 0.6  | 0.6  | 0.6  | 0.6  | 0.4  | 0.4  | 0.2  | 0    |
| FPR      | 1    | 1    | 0.8  | 0.8  | 0.6  | 0.4  | 0.2  | 0.2  | 0    | 0    | 0    |

La curva viene costruita considerando la probabilità, generata a posteriori dal modello,  $P(+|A)$  (che esso appartenga alla classe +) per ogni record e ordinandoli in base a questo valore in ordine decrescente.

Per ognuno dei record viene plottato un punto ( $FPR, TPR$ ), e l'insieme dei punti rappresenta la curva.



Normalmente nessun modello è genericamente meglio degli altri, ad esempio il modello  $M_1$  funziona bene quando  $FPR$  è piccolo (vogliamo evitare falsi positivi) mentre  $M_2$  funziona bene per  $FPR$  grandi (vogliamo evitare falsi negativi).



## 11. Clustering

Il **clustering** consiste nell'identificare gruppi di oggetti in modo tale che gli oggetti in un gruppo sono simili fra di loro e diversi dagli oggetti negli altri gruppi, può essere utilizzato per:

- **Capire** meglio i dati che si hanno davanti
- Ridurre la dimensione di dataset grandi, **sintetizzandoli**

### Tipi di clustering

Il clustering rappresenta un insieme di cluster e può essere di due tipi:

- **Partizionale**: gli oggetti vengono divisi in subset non sovrapposti in modo tale che ogni oggetto sia in un unico cluster
- **Gerarchico**: gli oggetti sono divisi in cluster nidificati organizzati in un albero gerarchico

Altre caratteristiche di un cluster possono essere:

- **Esclusività (cluster esclusivi/ non esclusivi)**: un punto non può appartenere a diversi cluster
- **Fuzziness (cluster fuzzy/ non fuzzy)**: ogni punto appartiene a tutti i cluster con un peso compreso tra 0 e 1 per ogni cluster
- **Parziale-Completo**: un algoritmo si dice parziale quando clusterizza solo una parte di dati, considerando il resto rumore
- **Eterogeno-Omogeneo**: i cluster creati hanno differenti o uguali dimensioni, forme e densità

I cluster possono essere di diversi tipi:

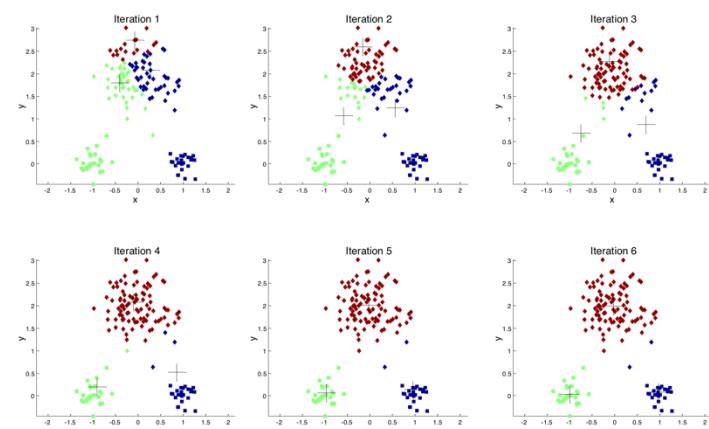
- **Well separated**: un cluster è un insieme di punti tale che ognuno dei punti in un cluster è più vicino a tutti gli altri punti del cluster rispetto a tutti quelli fuori dal cluster
  - **Center-based**: un cluster è un insieme di oggetti tali che un oggetto in un cluster è più vicino al centro del proprio cluster, piuttosto che al centro di un qualsiasi altro cluster, il centro può essere rappresentato da un **centroide** (l'effettiva media aritmetica degli oggetti nel cluster che spesso è un oggetto non esistente) o da un **medoide** (oggetto del dataset più vicino al centroide)
  - **Contiguity-based**: un cluster è un insieme di punti tali che un punto in un cluster sia più vicino a uno o più altri punti nel cluster che a qualsiasi altro punto che non è nel cluster
  - **Density-based**: un cluster è una regione densa di punti, separati da zone a bassa densità, usato quando i cluster sono irregolari e in presenza di rumore e outliers
  - **Cluster concettuali o Proprietà condivise**: individua cluster che condividono proprietà comuni o rappresentano un concetto in particolare
- 

### Algoritmi di clustering

#### K-means

L'algoritmo di clustering **K-means** è un algoritmo partizionale che associa ognuno dei  $k$  cluster ad un centroide (o medoide), ogni punto viene assegnato al cluster con il centroide più vicino.

Dopo la **selezione iniziale** dei  $k$  centroidi che avviene solitamente in maniera casuale, vengono creati i  $k$  cluster assegnando i punti al centroide più vicino e viene ricalcolata la posizione di tutti i centroidi rispetto ai punti dei vari cluster. Quest'ultima operazione viene ripetuta fino a che i centroidi smettono di spostarsi.



Un posizionamento dei centroidi iniziali **diverso**, genererà un risultato diverso e in alcuni casi peggiore.

La **distanza** è misurata solitamente mediante la distanza Euclidea, la cosine similarity, la correlazione etc.

La **convergenza maggiore** avviene durante le prime iterazioni.

La **complessità** dell'algoritmo è  $O(n_{points} * n_{clusters} * n_{iterations} * n_{attributes})$ .

La misura più comune per individuare quanto è efficace un certo clustering è **SSE** (Sum of Squared Error). Per ogni punto l'errore viene calcolato come la **distanza dal cluster** più vicino.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist^2(m_i, x)$$

In cui  $x$  rappresenta un punto nel cluster  $C_i$  e  $m_i$  rappresenta il centroide di  $C_i$ .

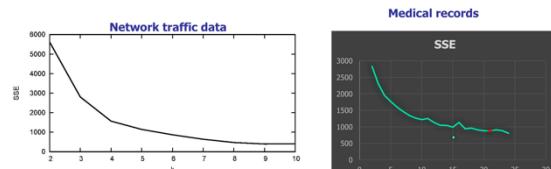
Più SSE è **grande**, peggiore è il clustering che abbiamo ottenuto.

Si nota che SSE tende ad assumere **valori decrescenti** all'aumentare di  $k$ , è perciò più efficace confrontare clustering con  $k$  uguale.

La **scelta dei centroidi iniziali** diventa cruciale in quanto influenza radicalmente la qualità del clustering ottenuto, questa scelta può avvenire in diversi modi:

- **Run multipli**: il clustering viene eseguito diverse volte e viene scelto quello con un SSE minore
- **Soluzione mista**: viene fatto un sample e in seguito viene applicato un algoritmo gerarchico per determinare i centroidi iniziali
- **Selezione dei centroidi distanti**: vengono generati più di  $k$  centroidi iniziali e vengono selezionati i più distanti
- **Postprocessing**: viene analizzato il risultato ottenuto al termine di un run per migliorare la posizione dei centroidi iniziali per il prossimo run
- **Bisezionamento** (trattato in seguito)

La selezione del  $k$  iniziale viene fatto analizzando l'**elbow graph** e identificando il ginocchio della curva. Questo grafico plotta l'SSE rispetto a  $k$ , viene scelto il  $k$  per cui il guadagno ottenuto dall'aggiunta di un centroide è trascurabile e la riduzione della qualità della misura non è più abbastanza interessante.



Può succedere che ad un certo punto nella curva l'SSE inizi ad aumentare, ciò rappresenta il fatto che vengono riconosciuti dei cluster vuoti che portano il numero di cluster realmente analizzati a diminuire rispetto al numero totale, ciò comporta una distanza maggiore e di conseguenza un errore maggiore (perché i punti sono distanziati quanto lo erano con un  $k$  minore).

Per limitare la presenza di cluster vuoti esistono diverse strategie per assegnare in maniera più efficace la posizione iniziale dei centroidi:

- Scegliere un punto dal cluster con SSE maggiore (punto più lontano di tutti)
- Scegliere il punto che influisce più di tutti sull'SSE

Il **pre-processing** dei dati per l'algoritmo k-means consiste nel:

- Normalizzare i dati
- Eliminare gli **outliers**

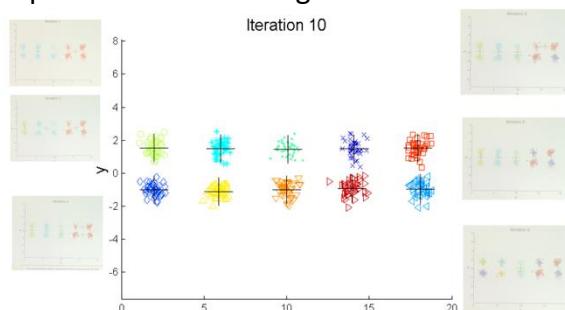
Il **post-processing** nell'algoritmo k-means consiste nel:

- Eliminare **piccoli** cluster che possono rappresentare **outliers**
- **Dividere** cluster con un SSE alto
- **Unire** cluster vicini con un SSE relativamente basso

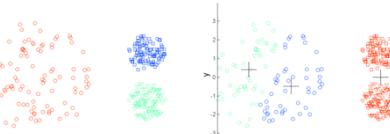
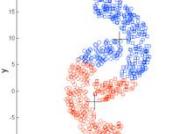
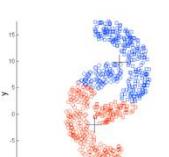
È possibile usare gli step relativi al post-processing anche durante il processo di clustering.

### Bisectiong K-means

Il **bisectiong k-means** consiste nel dividere, di volta un volta, in due il cluster con SSE più alto. Consiste nel eseguire ricorsivamente un k-means con  $k = 2$  e può portare a soluzioni generalmente migliori rispetto al k-means normale.

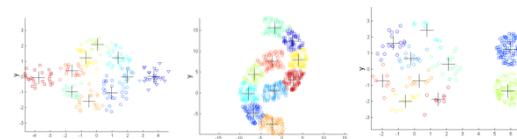


L'algoritmo k-means è problematico quando i cluster hanno:

| Dimensioni differenti                                                                                | Densità differente                                                                                         | Forma non globulare                                                                                    |                                                                                                             |                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <br>Original Points | <br>K-means (3 Clusters) | <br>Original Points | <br>K-means (3 Clusters) | <br> |

Inoltre il k-means è problematico in presenza di outliers.

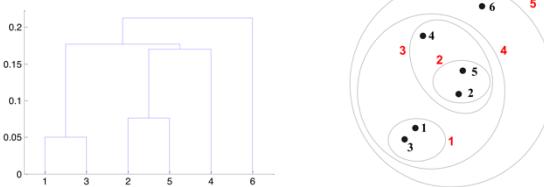
Una soluzione per superare i limiti descritti è quella di usare **diversi** cluster che poi verranno uniti fra di loro nella fasi di **postprocessing**.



### Clustering gerarchico

Il clustering **gerarchico** produce un insieme di cluster nidificati organizzati in un albero gerarchico e rappresentati mediante un **dendogramma**.

Scansionando il dendogramma dall'alto verso il basso vengono individuate tutte le possibili soluzioni, a partire da quella contenente un solo cluster fino a quella completamente frammentata.



**Tagliare** il dendogramma consiste nello scegliere uno dei clustering, ogni intersezione rappresenta un cluster.

Uno dei punti di **forza** del clustering gerarchico consiste nel non avere particolari parametri in ingresso, in quanto può essere ottenuto qualsiasi numero di cluster.

Un altro punto di forza risiede nel fatto che i cluster possono corrispondere a effettive relazioni gerarchiche. (animale={mammifero={cane, gatto, ...}, rettile, ...})

È possibile distinguere due tipi di clustering gerarchico:

- **Agglomerativo**: partendo dai punti considerati come cluster individuali, ad ogni step viene unita la coppia di cluster più vicina, fino a che non ne rimane un solo cluster
- **Divisivo**: a partire da un unico cluster, esso viene diviso di volta in volta fino a quando ogni punto si trova in un cluster

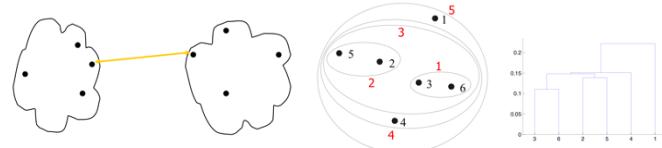
Gli algoritmi gerarchici tradizionali utilizzano una **matrice** delle distanze o delle similarità.

Gli algoritmi gerarchici più popolari sono quelli **agglomerativi**, l'algoritmo di base funziona in questo modo:

- Viene calcolata la matrice di prossimità
- Si assegna ogni punto ad un cluster diverso
- Si ripetono queste operazioni fino a che non rimane un solo cluster:
  - Unire i due cluster più vicini
  - Aggiornare la matrice delle prossimità (eliminare le righe e colonne dei cluster uniti e aggiornando i valori di distanza)

L'operazione chiave è quella che calcola la prossimità fra due cluster, considerando le coppie di punti tali che un punto sia del primo cluster e l'altro del secondo, esistono diversi approcci:

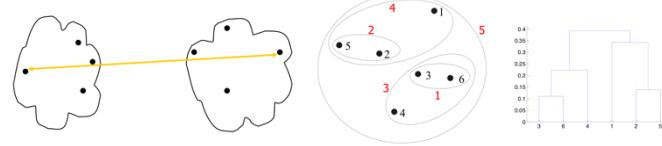
- **Single link (o MIN)**: viene scelta come distanza fra i due cluster la distanza minima fra le possibili coppie.



Il punto di **forza** è che MIN riesce a identificare cluster con **granularità diversa** e identifica anche cluster di forma non globulare.

Il punto di **debolezza** è che MIN rappresenta male i punti di contatto fra i gruppi, ignorando la presenza di dati rumorosi nel mezzo.

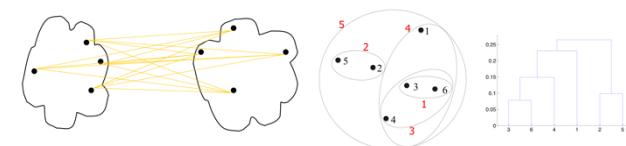
- **Complete linkage (o MAX)**: viene scelta come distanza fra i due cluster la distanza massima fra le possibili coppie.



Il punto di **forza** è che MAX si comporta bene in presenza di dati rumorosi e outliers.

Il punto di **debolezza** è che MAX tende a rompere i cluster grandi, e identifica spesso cluster di forma globulare.

- **Group average**: la distanza fra i due cluster è rappresentata dalla media di tutte le distanze delle possibili coppie di gruppi.



Il punto di **forza** e di **debolezza** sono simili a quelli del MAX.

- **Distanza fra i centroidi:** la distanza fra due cluster è la distanza fra i due centroidi, in caso di gruppi e punti la distanza considerata è quella fra il centroide e il punto.

Il costo in termini di **spazio** è  $O(N^2)$ , considerando  $N$  il numero di punti, in quanto deve essere memorizzata una matrice di prossimità.

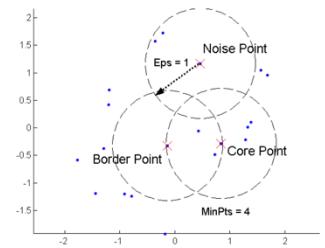
Il costo in termini di **tempo** è  $O(N^3)$ , considerando  $N$  il numero di punti, perché ci sono  $N$  step e ad ogni step viene aggiornata la matrice di prossimità  $N * N$ .

### DBSCAN

Il **DBSCAN** è un algoritmo density-based che richiede due variabili in ingresso **Eps** e **MinPts**, assegna ogni punto a una di queste tre categorie:

- **Core point:** ha più di MinPts punti all'interno della circonferenza di raggio Eps
- **Border point:** ha meno di MinPts all'interno della circonferenza di raggio Eps, ma è nelle vicinanze di un core point
- **Noise point:** è qualsiasi punto che non sia core o border.

Mediante questa suddivisione è in grado di identificare rumore e outliers.



Dopo aver eliminato i **noise points**, viene effettuato il clustering per tutti i punti rimanenti.

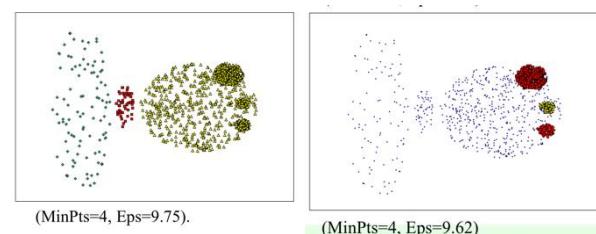
Quindi per tutti i punti core:

- Se il core point in questione non appartiene ad un cluster, gliene viene assegnato uno nuovo
- Questo nuovo cluster viene assegnato anche a tutti i punti nel vicinato non ancora etichettati (nello specifico se due cluster si trovano a una distanza minore di Eps, essi vengono uniti).



Il DBSCAN è quindi **resistente al rumore** e agli **outliers** e riesce a gestire cluster di dimensioni e forme diverse.

Non funziona bene in presenza di punti a **densità variabile** e per dati con molte dimensioni.



La soluzione al problema della densità è il **multiple level clustering** che consiste nell'eseguire diversi run, nei primi vengono considerati i cluster a densità più alta (eps minore) considerando il resto come rumore, in seguito si vanno a caratterizzare i cluster meno densi.

La difficoltà maggiore del DBSCAN è nella scelta dei **parametri iniziali**.

Dobbiamo considerare, durante la scelta dei parametri, che **MinPts** rappresenterà il numero minimo di punti che potremo avere in un cluster e per questo diventa più facile da identificare.

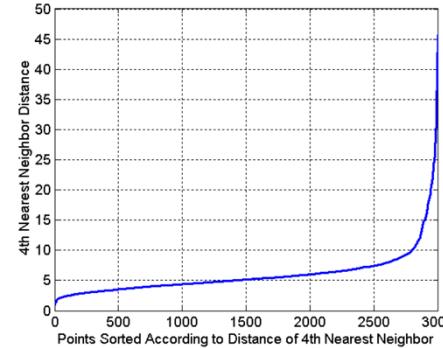
La scelta di **Eps** invece viene fatta secondo il metodo **k-dist** che consiste nel definire un  $k$  e analizzare la distribuzione delle distanze fra ogni punto e il suo  $k$ -esimo vicino.

Se è stato già scelto un MinPts,  $k$  assumerà questo valore.

Considerando queste distanze è possibile costruire un grafico, ordinando i punti per distanza crescente, che permette di ottenere informazioni preventive su come i dati sono distribuiti.

È consigliabile effettuare la scelta di Eps, e quindi il taglio della curva, in corrispondenza del gomito.

In questo caso il taglio ideale è a 10, per cui avremo circa 2800 core points.



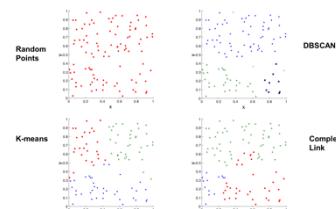
## Validazione degli algoritmi di clustering

Per verificare la **validità** di un algoritmo di clustering abbiamo la necessità di confrontare risultati diversi ottenuti da algoritmi diversi, o quelli ottenuti dallo stesso algoritmo ma con parametri diversi.

Possiamo analizzare il clustering ottenuto in diversi modi:

- Un diagramma di dispersione dei centroidi
- Un **radar graph** che rappresenta i centroidi e i rispettivi cluster sulle punte
- Un **boxplot** per ogni feature che mostra l'insieme dei valori all'interno di ogni gruppo
- L'analisi di **variabili esterne** non analizzate durante la fase di clustering

Dato un set di punti random si può analizzare il comportamento dei vari algoritmi. Solo il DB scan trova che la maggior parte dei gruppi sono rumorosi. Altri algoritmi trovano cluster diversi.



### Indici di validazione

Gli indici per la validazione possono essere di tre tipi:

- **Indici interni:** usati per misurare la bontà del clustering sfruttando solo la conoscenza contestuale al clustering, senza l'utilizzo di dati aggiuntivi (SEE, coesione, separazione, (adjusted) Rand-index, Silhouette):
  - La **Cluster Cohesion** misura quanto sono strettamente correlati gli oggetti in un cluster e si misura come:

$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

Nello specifico calcola la distanza fra un punto  $x$  ed il suo centroide  $m_i$  al quadrato, per ogni elemento nel cluster  $C_i$ , per ogni cluster  $i$ .

- La **Cluster Separation** misura quanto un cluster è distinto dagli altri cluster e si misura come:

$$BSS = \sum_i |C_i| (m - m_i)^2$$

Nello specifico calcola la distanza fra il valore medio dell'intero dataset  $m$  e il centroide del singolo cluster  $m_i$  al quadrato, pesata per la cardinalità del cluster  $|C_i|$ , per ognuno dei cluster  $i$ .

- La **Silhouette** è una misura (può essere plottata in una curva) che utilizza i concetti di coesione e separazione che misura quanto ogni oggetto si trova bene nel proprio cluster.

Può essere definita per:

- Punti individuali
- Cluster individuali
- L'intero clustering

Le ultime due rappresentano un'aggregazione della misura per punti individuali, calcolando il valore medio.

Si calcola come  $s(i) = \frac{b(i)-a(i)}{\max\{a(i), b(i)\}}$

In cui:

- $a(i)$  rappresenta la dissimilarità media dell'oggetto  $i$  con tutti gli altri oggetti del suo cluster (sintesi della coesione)
- $b(i)$  rappresenta il valore minimo fra i valori medi di dissimilarità dell'oggetto  $i$  con gli altri oggetti presenti negli altri cluster di cui  $i$  non fa parte (sintesi della separazione)

I valori di Silhouette variano fra -1 e +1, in cui +1 è il valore ideale.

- Il **Rand-index** invece consente di valutare quanto 2 partizionamenti ottenuti con algoritmi diversi (o configurazione iniziale diversa) sono simili o dissimili. Può anche valutare la bontà di un algoritmo di clustering conoscendo le etichette di classe effettive.

Si calcola come  $\text{Rand Index} = \frac{f_{00}+f_{11}}{f_{00}+f_{01}+f_{10}+f_{11}}$

Con  $f_{ij}$  il numero di coppie di oggetti che (non) appartengono alla stessa classe se  $i$  è 1 (0), (non) appartengono allo stesso cluster se  $j$  è 1 (0).

Al posto di classe – cluster si può fare l'analisi per clustering1 – clustering 2

Consideriamo due partizionamenti di esempio:

| P1 |    |    |    |
|----|----|----|----|
| O1 | O2 | O3 | C1 |
| O4 | O5 | O6 | C2 |

| P2 |    |    |    |
|----|----|----|----|
| O1 | O2 | O3 | C1 |
| O4 | O5 | O6 | C2 |

Consideriamo tutte le possibili combinazioni di oggetti

O1,O2→In entrambi i partizionamenti sono in C1→  $f_{11}$

O1,O4→Sono in cluster diversi in P1 e nello stesso cluster in P2→  $f_{01}$

...

O2,O3→Sono nello stesso cluster per P1 e in cluster diversi per P2→  $f_{10}$

...

O2,O5→In entrambi i partizionamenti si trovano in cluster diversi →  $f_{00}$

...

Per confrontare molteplici algoritmi si genera una matrice di random index con tutte le possibili combinazioni

- **Indici esterni:** usati per misurare quanto le etichette del clustering corrispondono a etichette di classe note a posteriori (**entropia e purezza**):
  - **Entropia:** vogliamo che sia minimizzata
  - **Purezza:** vogliamo che sia massimizzata
- **Indici relativi:** usati per comparare due clustering o due cluster

Questi indici sono utili per una prima validazione ma vanno corredati da informazioni aggiuntive che caratterizzano i singoli cluster ottenuti.

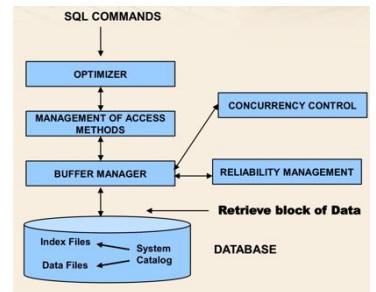
## 12. Introduzione ai DBMS

Il **DBMS** (Data Base Management System) è un software progettato per memorizzare e gestire i database.

### *Componenti di un DBMS*

Un DBMS è formato da diversi componenti:

- **Ottimizzatore:**
    - Riceve in input la query SQL eseguendo un analisi lessicale, sintattica e semantica rilevando eventuali errori
    - Rappresenta la query mediante l'algebra relazionale (procedurale)
    - Sceglie un **piano di esecuzione** per l'accesso ai dati e l'esecuzione delle operazioni basandosi su dati statistici relativi alla distribuzione dei dati
  - Questo componente garantisce l'**indipendenza dei dati** all'interno del modello relazionale, in quanto la forma con cui la query SQL viene scritta non influisce nel modo in cui il risultato viene calcolato, inoltre una riorganizzazione fisica non richiede la riscrittura delle query.
  - **Access Method Manager:** effettua l'accesso ai dati utilizzando la strategia selezionata dall'ottimizzatore
  - **Buffer Manager:** gestisce l'area riservata al DBMS nella memoria principale, utilizzata per gestire le operazioni di lettura e scrittura delle **pagine** (unità di informazione in un db). Il blocco di memoria in questione è **preallocato** ed è condiviso fra le diverse applicazioni che hanno accesso al database.
  - **Concurrency Control:** gestisce l'accesso concorrente ai dati garantendo che le applicazioni non interferiscono l'una con l'altra.
  - **Reliability Manager:** garantisce la correttezza del contenuto del database in caso di **crash**, garantendo l'esecuzione **atomica** delle transazioni.
- Per far ciò sfrutta strutture ausiliari (log) che lo aiutano a recuperare lo stato corretto del db dopo una failure.



### Transazioni

Una **transazione** è l'unità logica di lavoro eseguibile da un'applicazione, rappresenta un sequenza di una o più istruzioni SQL che eseguono operazioni di lettura e scrittura nel db, è caratterizzata da:

- Correttezza
- Affidabilità
- Isolamento

Una nuova transazione inizia implicitamente quando viene riconosciuto il primo comando SQL e può terminare in due modi:

- **COMMIT**: la transazione è stata eseguita correttamente
  - **ROLLBACK**: la transazione non è stata eseguita per un errore e il db è stato ripristinato alla situazione iniziale (prima dell'inizio della transazione).
- Può essere richiesto dalla transazione stessa (suicide) o dal sistema (murder)

## ACID

Una transazione è caratterizzata dalle proprietà **ACID**:

- **Atomicity**: una transazione non può essere divisa in unità più piccole e non è possibile lasciare il database in uno stato intermedio di esecuzione.  
È garantita dalle primitive:
  - **Undo**: il sistema disfa le operazioni effettuate dalla transazione fino al punto corrente (usato per il rollback)
  - **Redo**: il sistema riesegue le operazioni eseguite dalle transazioni committute (usato in caso di failure)
- **Consistency**: l'esecuzione di una transazione non deve violare l'integrità del database, quando viene identificata una violazione il sistema deve poter eseguire il rollback dell'ultima transazione
- **Isolation**: l'esecuzione di una transazione deve essere indipendente dall'esecuzione corrente di altre transazioni
- **Durability**: l'effetto di una transazione committata non deve essere perso nel tempo (garantita dal Reliability Manager)

## 13. Buffer Manager

Il **buffer manager** gestisce il trasferimento delle pagine dal disco alla memoria e viceversa, sovrintendendo alle operazioni relative al DBMS buffer.

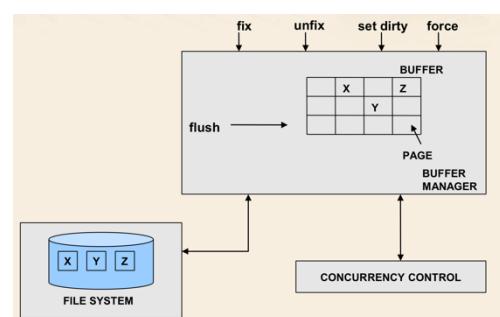
Il **buffer** è un blocco di memoria preallocato al DBMS condiviso fra tutte le transazioni in esecuzione. Esso è organizzato in **pagine** la cui dimensione dipende dalla dimensione del blocco I/O del sistema operativo.

Utilizza delle strategie di gestione della memoria:

- **Località temporale**: è più probabile che i dati utilizzati di recente vengano utilizzati di nuovo
- **Legge empirica 20-80**: il 20% dei dati sono letti/scritti dall'80% delle transazioni

Il buffer manager mantiene anche fotografie addizionali del contenuto corrente del buffer, per ogni pagina del buffer:

- La **locazione fisica** della pagina nel disco: identificatore del file e il numero del blocco
- **Variabili di stato**:
  - **Count**: numero di transazioni che stanno utilizzando una certa pagina
  - **Dirty bit**: che mostra se la pagina è stata modificata



## *Primitive del buffer manager*

Il buffer manager fornisce diverse primitive per accedere ai metodi che caricano o memorizzano le pagine sul disco:

- **Fix:** usata dalla transazioni per **richiedere accesso ad una pagina** sul disco:
  - La pagina richiesta viene cercata all'interno del buffer
  - Se la pagina è nel buffer, ne viene incrementato il count
  - Se la pagina non viene trovata deve essere identificata una posizione nel buffer in cui caricare la nuova pagina:
  - Se non vengono trovate pagine libere, viene selezionata la prima pagina libera che ha count=0, se essa ha anche dirty=1 viene scritta sul disco
  - L'indirizzo della pagina all'interno del buffer viene restituito alla transazione che ne ha fatto richiesta.
- **Unfix:** comunica al buffer manager che la transazione **non sta più usando una certa pagina**, di conseguenza il buffer manager decrementerà il count di essa
- **Set dirty:** comunica al buffer manager che **la transazione ha modificato la pagina richiesta**, di conseguenza il buffer manager setterà il dirty bit a 1
- **Force:** richiede il **trasferimento sincrono di una pagina nel disco**, di conseguenza la transazione in esecuzione viene sospesa fino a che la primitiva non è stata eseguita, provoca sempre una scrittura su disco
- **Flush:** consiste nel trasferimento di una pagina su disco, indipendentemente dalle richieste effettuate dalle transazioni. È una primitiva utilizzata internamente dal buffer manager, mentre la CPU è in idle, per salvare le pagine con count=0 e quelle che non vengono utilizzate da tempo.

## *Strategie di scrittura nel buffer*

Il buffer manager ha diverse **strategie** di scrittura:

- **Steal:** il buffer manager può selezionare una pagina bloccata (lock, che appartiene ad una transazione) con count=0 quando ha bisogno di uno slot libero.  
La politica **steal** salva su disco pagine **dirty** che appartengono a transazioni **non committate**, in caso di failure bisogna poter recuperare lo stato precedente mediante un rollback.
- **No steal:** il buffer manager non può selezionare pagine bloccate (lock) da transazioni attive quando ha bisogno di uno slot libero
- **Force:** quando viene effettuata una operazione di commit, il buffer manager scrive su disco, in maniera sincrona, tutte le pagine attive di una transazione
- **No force:** dopo una commit, le pagine vengono scritte asincronicamente (mediante la primitiva di flush).  
Le pagine che appartengono a transazioni **committate** potrebbero venir scritte su disco **dopo la commit**, in caso di **failure** prima del salvataggio su disco questi cambiamenti devono poter essere riapplicati mediante un'azione di **redo** (grazie a file di log).

Tipicamente la configurazione utilizzata è **steal** e **no force** che offre le performance migliori:

- No force offre migliori performance I/O
- Steal è una scelta obbligata per query che accedono a tante pagine contemporaneamente

## *Buffer manager e filesystem*

Il Buffer Manager sfrutta diversi servizi forniti direttamente dal filesystem:

- **Creation/deletion of a file**

- **Open/close of a file**
- **Read:** offre un accesso diretto ai blocchi di un file e richiede l'identificativo del file, il numero di blocco e la pagina del buffer in cui salvare i dati presi dalla memoria
- **Sequential read:** offre l'accesso sequenziale ad un numero prefissato di blocchi in un file e richiede l'identificativo del file, il blocco di partenza, il numero di blocchi che devono essere letti e la pagina iniziale del buffer in cui salvare i dati presi dalla memoria
- **Write and Sequential Write:** analogo alla lettura ma per il salvataggio dei dati su disco
- Funzioni di gestione delle cartelle

## 14. Accesso fisico ai dati

I dati possono essere storicizzati sul disco mediante **diverse strutture** e formati che permettono di avere un'esecuzione delle **query efficiente**. Le strutture di accesso fisico descrivono come i dati vengono memorizzati sul disco.

L'**Access Method Manager** trasforma il piano di accesso generato dall'ottimizzatore in una sequenza di richieste di accesso fisico alle pagine del disco (database) utilizzando diversi **metodi di accesso**.

### *Metodi di accesso*

Un **metodo di accesso** è un modulo software specializzato per una singola tipologia di struttura fisica e offre primitive di **lettura e scrittura**.

I **metodi di accesso** selezionano il blocco di un file (che poi verrà caricato mediante le primitive del buffer manager) che deve essere caricato in memoria avendo una conoscenza a priori della distribuzione dei dati (e delle tuple) nei diversi file.

### *Organizzazione di una pagina*

Una pagina all'interno del disco è composta da:

- Spazio per la memorizzazione dei dati
- Spazio riservato alle informazioni di controllo relative ai metodi di accesso (metadati), in quanto la struttura della pagina cambia, se cambia il metodo di accesso utilizzato
- Spazio riservato alle informazioni di controllo relative al file system

Le tuple possono avere dimensioni diverse sia per via dei tipi di **varchar** contenuti che per la presenza di **valori nulli**.

Una singola tupla può occupare una sola pagina o diverse pagine.

### *Strutture di accesso fisico*

Le **strutture di accesso fisico** descrivono come i dati vengono memorizzati nel disco per predisporvi ad una esecuzione efficiente delle query, possono essere di due tipi:

- **Primarie/Clustered:** memorizzano i dati effettivi del database relativi alle tuple
- **Secondarie/Non Clustered:** non memorizzano l'informazione completa ma solo un sottosinsieme di dati della tupla e il puntatore alla tupla effettiva in memoria

Le diverse strutture di accesso fisico sono:

- **Physical data storage:** contengono l'effettivo contenuto informativo
  - **Strutture sequenziali:** le tuple vengono memorizzate nella pagina in maniera sequenziale secondo diversi criteri:

- **Heap file:** le tuple vengono messe in sequenza in base all'ordine di inserimento che si comporta come un semplice append, ciò rende la lettura e la scrittura sequenziali efficienti.

In questo modo il blocco viene completamente riempito prima di passare al successivo.

Le operazioni di **delete** o **update** però possono creare uno spreco di spazio (una delete può lasciare dello spazio vuoto, mentre una update può ridimensionare una tupla in modo che debba essere allocata in un nuovo spazio di memoria).

Viene frequentemente usato in **DBMS relazionali** assieme ad un indice unclustered per il supporto alle operazioni di ricerca e di ordinamento.

- **Struttura sequenziale ordinata:** l'ordine di scrittura delle tuple dipende da un chiave formata da uno o più attributi, chiamata **sort key** (potrebbe essere la chiave primaria stessa).

È utile per:

- Operazioni di **ordinamento** e **group by** rispetto alla sort key
- Operazioni di **ricerca** sulla sort key
- Operazioni di **join** sulla sort key (merge join)

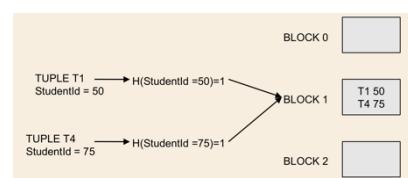
Una delle criticità è quella di preservare l'ordinamento quando vengono inseriti nuovi valori, per limitare questo problema è possibile operare in due modi:

- **Lasciare libera una percentuale** di spazio all'interno di ogni blocco durante la creazione della tabella (in fase di inserimento e cancellazione vengono riordinate dinamicamente le tuple nel singolo blocco)
- Vengono predisposti dei file (di **overflow**) appositi che conterranno le tuple che non entrano nel blocco corretto

Questa tipologia di strutture sequenziali **utilizza indici B<sup>+</sup>-Tree clustered** e viene utilizzata dal DBMS per memorizzare risultati intermedi.

- **Strutture hash:** garantiscono un accesso diretto ed efficiente ai dati basato completamente sul valore della chiave (che include uno o più attributi).

La **posizione del record** viene determinata mediante un **hash della chiave** che avrà un valore compreso tra 0 e  $B - 1$  (con  $B$  il numero di blocchi), che determina il blocco in cui inserire/ricercare la tupla.



Per consentire l'inserimento di nuovi dati, i blocchi non devono mai essere riempiuti completamente.

Vantaggi:

- Efficiente per query con predicati di ugualanza sulla chiave
- Non è richiesto alcun ordinamento

Svantaggi:

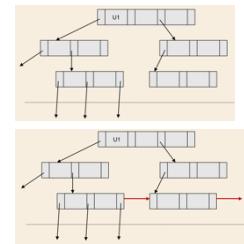
- Inefficiente per query con **predicati range**
- Possono avvenire delle collisioni

- **Indici:** utili ad incrementare l'efficienza di accesso (velocità I/O)
  - **Strutture ad albero** (B-Tree, B<sup>+</sup>-Tree): molto utilizzate nei DBMS relazionali offrono un accesso diretto ai dati basato sul valore di un **campo chiave** (contenente uno o più attributi), la posizione fisica delle tuple non è vincolata.  
Un albero è solitamente composta da un **nodo radice**, ogni nodo ha diversi **figli** e i **nodi foglia** forniscono l'accesso effettivo ai dati.

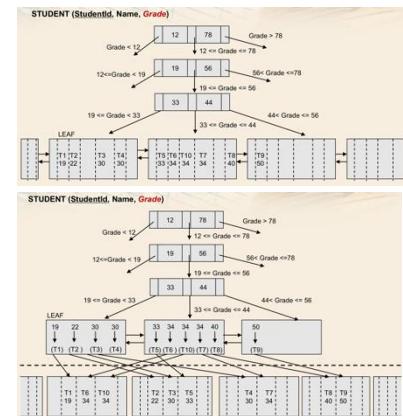
Esistono due differenti strutture ad albero per l'indicizzazione:

- **B-Tree:** le pagine possono essere raggiunte solo visitando l'albero a partire dal nodo radice
- **B<sup>+</sup>-Tree:** fornisce una struttura di collegamenti che consente un accesso sequenziale, ordinato in base ai valori chiave

In entrambi i casi **B** sta per **bilanciato** in quanto ogni foglia è alla stessa distanza dal nodo radice e i tempi di accesso sono sempre gli stessi per qualsiasi valore ricercato.



Per un utilizzo dell'albero di tipo **clustered** l'intero contenuto della tupla è all'interno dei nodi foglia, viene impiegato per indexing rispetto alla **primary key**.



Per un utilizzo dell'albero di tipo **unclustered** i nodi foglia contengono un sottoinsieme del contenuto informativo della tupla e un puntatore alla tupla in memoria, viene impiegato per indici secondari.

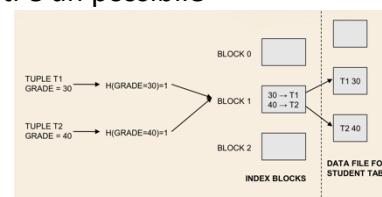
Vantaggi:

- Efficiente per query con **predicati range**
- Efficiente per **scan sequenziale** in ordine di campo chiave

Svantaggi:

- Gli inserimenti potrebbero richiedere lo split di una foglia, che richiede della computazione
- Le eliminazioni potrebbero richiedere l'unione di nodi vuoti e un possibile ribilanciamento

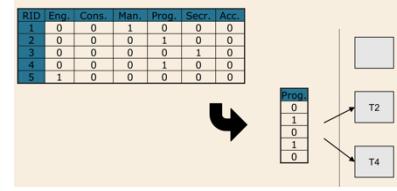
- **Unclustered hash index:** si differenzia dalle strutture hash semplici perché è unclustered, il blocco contiene solo un sottoinsieme delle informazioni della tupla e un puntatore alla tupla in memoria.
- **Bitmap index:** anch'esso utilizza un campo chiave e si basa su una **matrice di bit**. Questa matrice riferisce i record mediante il **RID** (Row IDentifier) e ha una colonna per ogni possibile valore diverso del campo chiave. Una cella viene valorizzata a 1 se la tupla *i* ha valore *j*.



| RID | Val <sub>1</sub> | Val <sub>2</sub> | ... | Val <sub>n</sub> |
|-----|------------------|------------------|-----|------------------|
| 1   | 0                | 0                | ... | 1                |
| 2   | 0                | 0                | ... | 0                |
| 3   | 0                | 0                | ... | 1                |
| 4   | 1                | 0                | ... | 0                |
| 5   | 0                | 1                | ... | 0                |

Vantaggi:

- Efficiente per query con **espressioni booleane di predici** e per **attributi categorici** (basta leggere una colonna della tabella)
- Efficiente per il **bitmapped join**
- Appropriato per attributi con un dominio di valori limitato



Svantaggi:

- Non è utilizzabile per valori continui
- Lo spazio necessario cresce significativamente all'aumentare della cardinalità dei valori dell'attributo

## 15. Progettazione fisica

L'obiettivo della **progettazione fisica** è quello di definire strutture dati per l'ottimizzazione dell'esecuzione delle query.

La progettazione fisica non influisce in alcun modo sui risultati delle query svolte dalle applicazioni ed è completamente indipendente dalla progettazione logica.

Prima di effettuare la progettazione fisica è necessario:

- Definire lo schema logico
- Individuare il DBMS utilizzato e la lista delle sue feature
- Analizzare il carico di lavoro
  - Le **query più importanti** e la loro frequenza stimata
  - Le **operazioni di update** e la loro frequenza stimata
  - Le **performance richieste** per le query rilevanti e per gli update

Mentre la progettazione logica è difficile che venga modificata nel corso del tempo, la **progettazione fisica deve cambiare** per adattarsi alla situazione corrente.

La fase di progettazione fisica deve fornire in **output**:

- L'organizzazione delle **tabelle**
- Gli **indici** da utilizzare
- I **parametri di setup** per lo storage e per il DBMS

### Tipi di strutture fisiche

Esistono diversi tipi di **strutture fisica**:

- Non ordinate (heap)
- Ordinate (clustered)
- Hashing
- Insiemi di più relazioni (le tuple appartenenti a tabelle diverse possono essere interacciate)

### Caratteristiche del carico di lavoro

La caratterizzazione del carico di lavoro deve avvenire:

- Per ogni **query**:
  - Tabelle utilizzate
  - Attributi visualizzati
  - Attributi utilizzati in operazioni di selezione e di join

- Selettività della selezione
- Per ogni **update**:
  - Gli attributi e le tabelle coinvolte nella selezione
  - Selettività della selezione
  - Il tipo di update (Insert, Delete, Update) e gli attributi coinvolti

In seguito alla caratterizzazione del carico di lavoro vengono **selezionate le strutture fisiche** da utilizzare per i **dati** e per gli **indici**. Per ogni **tabella** va individuata la **struttura dei file** (heap o clustered) e gli **attributi da indicizzare** (hash o B<sup>+</sup>-Tree, clustered o unclustered).

In caso di modifiche nel modello logico, potrebbe essere necessario modificare lo schema fisico.

### [Selezione delle strutture dati](#)

Esistono due criteri generali per la progettazione fisica:

- Visto che la **chiave primaria** è spesso utilizzata per le operazioni di selezione e di join, in questi casi è opportuno indicizzare la chiave primaria
- È opportuno aggiungere degli indici anche per le query più comuni, in questo è necessario:
  - Selezionare una query frequente
  - Considerare il suo piano di esecuzione corrente
  - Definire un nuovo indice e considerare il nuovo piano di esecuzione, se i tempi migliorano lo si mantiene
  - Verificare l'effetto del nuovo indice sul carico di lavoro e sulla memoria

**Mai** creare indici per:

- **Tabelle piccole**: il caricamento dell'intera tabella richiede poche letture
- Attributi con cardinalità di dominio bassa: sono poco selettivi (noi si applica ai DW in quanto possono essere implementati mediante indici bitmap)

Per gli attributi appartenenti a **predicati semplici** di una where:

- Predicati di ugualanza: hash (preferred) or B<sup>+</sup>-Tree
- Predicati range: B<sup>+</sup>-Tree

Per where che includono **molti predicati semplici** potrebbe essere conveniente utilizzare indici **composti** (multi attributo) selezionando l'**ordine** (prima il più selettivo) di chiavi appropriato e considerando che questo tipo di indici hanno un costo di **mantenimento** elevato.

### [Operazioni di Join](#)

Le operazioni di **join** possono essere eseguite mediante:

- **Nested loop**: è un metodo di join che funziona bene in presenza di tabelle sbilanciate, viene scansionata la tabella piccola N volte, con N la cardinalità della tabella più grande. In questo caso è opportuno indicizzare la **inner table**.
- **Merge scan**: il join viene effettuato in seguito all'ordinamento delle tabelle in base alle chiavi di join, è opportuno per due tabelle grandi e per tabelle già ordinate. In questo caso è conveniente utilizzare il B<sup>+</sup>-Tree clustered.

### [Operazioni di Group by](#)

Le operazioni di group by possono avvenire in due modi:

- Facendo l'**ordinamento** per generare i gruppi

- Attraverso una funzione di **hash**

Per migliorare le operazioni di **group by** potrebbe essere opportuno indicizzare gli attributi della group by mediante hash index o B<sup>+</sup>-Tree.

La group by inoltre può essere automaticamente anticipata (group by **push down**) dall'ottimizzatore prima delle operazioni di join, ciò consente di produrre un numero minore di tuple su cui fare il join.

Esempio:

```
PRODUCT (Prod#, PName, PType, PCategory)
SHOP (Shop#, City, Province, Region, State)
SALES (Prod#, Shop#, Date, Qty)
```

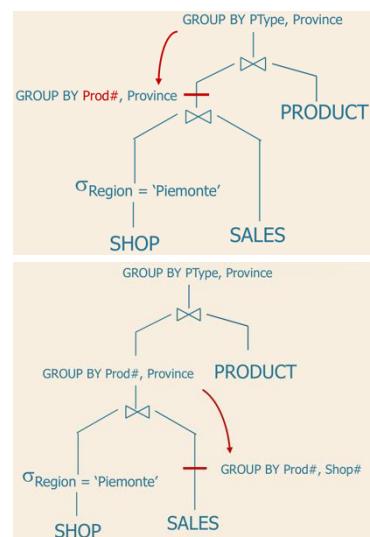
```
SELECT PType, Province, SUM (Qty)
FROM Sales S, Shop SH, Product P
WHERE S.Shop# = SH.Shop#
AND S.Prod# = P.Prod#
AND Region = 'Piemonte'
GROUP BY PType, Province;
```

Lo schema relazionale mostra l'ordine secondo il quale la query deve essere eseguita:

- Lettura delle tabelle
- Predicato di selezione (su regione) che riduce la cardinalità di shop
- Join fra shop e sales
- Join con product
- Group by
- Operatore di proiezione che seleziona solo i campi della select

Come si nota, la group by può essere anticipata per diminuire di volta in volta il volume di dati prima delle join.

In questo caso però non è possibile eliminare del tutto le group by superiori.



Quando ne abbiamo la possibilità è sempre utile **anticipare** le operazioni di group by rispetto a quelle di join.

### *Statistiche dell'ottimizzatore*

Quando l'ottimizzatore elabora il piano di esecuzione sfrutta delle **statistiche** che sono frutto dei **piani di esecuzione precedenti**.

Questi dati consentono di eseguire le query in modo più efficiente ma è necessario che vengano **aggiornati** periodicamente dall'utente.

### *Hints*

Mediante gli **hints** è possibile dare suggerimenti all'ottimizzatore rispetto agli indici da utilizzare, in questo caso l'ottimizzatore seguirà gli hints e non cercherà/attuerà la soluzione statisticamente migliore.

### *Esempi*

Esempi sulla creazione di indici dato questo schema logico:

- EMP (Emp#, EName, Dept#, Salary, Age, Hobby)
- DEPT (Dept#, DName, Mgr)
  - In EMP  
Dept# FOREIGN KEY REFERENCES DEPT.Dept#
  - In DEPT  
Mgr FOREIGN KEY REFERENCES EMP.Emp#

- In questo caso è opportuno inserire un indice sull'attributo Salary (B<sup>+</sup>-Tree). In questo caso l'indice potrebbe essere ignorato per via dell'espressione aritmetica.

```
SELECT *
FROM EMP
WHERE Salary/12 = 1500;
```

- Questa query è equivalente alla precedente ma è stata rimossa l'espressione aritmetica, in questo caso dobbiamo analizzare la selettività di questa condizione, considerando anche che avendo \* nella select, avremo bisogno di memorizzare le intere tuple all'interno dell'indice. Se i dipendenti che soddisfano questa condizione sono effettivamente pochi, allora la creazione dell'indice è conveniente.

```
SELECT *
FROM EMP
WHERE Salary = 18000;
```

- Supponiamo che la tabella EMP contiene 30 tuple per blocco:

- Card(DEPT)=50: in questo caso, utilizzando l'indice avrò bisogno di leggere 1-2 blocchi per la lettura dell'indice e un blocco per la lettura della tupla dalla memoria. Utilizzando un accesso sequenziale invece, dovrei leggere direttamente dalla memoria un **massimo di 2 blocchi**.

Si nota che in questo caso la creazione dell'indice **non conviene**.

- Card(DEPT)=2000: anche in questo caso, utilizzando l'indice avrò bisogno di leggere i blocchi contenuti l'indice e un blocco per la lettura della tupla dalla memoria. Utilizzando un accesso sequenziale, invece, potrei arrivare a leggere dalla memoria un **massimo di 70 blocchi**.

Si nota che la creazione dell'indice, in questo caso, è **conveniente** se il predicato è **selettivo**.

- In questo caso possiamo creare due indici

- Un hash index su DName per la condizione di selezione
- Un hash index su E.Emp#, che permette di ordinare la chiave di join per un nested loop con Emp che fa da tabella interna.

```
SELECT EName, Mgr
FROM EMP E, DEPT D
WHERE E.Emp# = D.Mgr
AND DName = 'Toys';
```

- In questo caso può essere conveniente un indice su Age, ma ciò dipende dalla selettività della condizione.

```
SELECT EName, Mgr
FROM EMP E, DEPT D
WHERE E.Emp# = D.Mgr
AND DName = 'Toys'
AND Age=25;
```

- Abbiamo due alternative relativa ai prediciati di selezione:

- Hash index su Hobby
- B<sup>+</sup>-Tree su Salary

```
SELECT EName, Mgr
FROM EMP E, DEPT D
WHERE E.Emp# = D.Mgr
AND Salary BETWEEN 10000 AND 12000
AND Hobby='Tennis';
```

In questo caso verrà considerato dall'ottimizzatore solo uno dei due indici, esistono anche degli ottimizzatori più avanzati che calcolano l'intersezione dei due indici prima di leggere le tuple.

Per quanto riguarda la join abbiamo due alternative:

- Hash join
- Nested loop: considerando DEPT come tabella interna in quanto i prediciati di selezione sono su EMP, usiamo anche un indice su DEPT.Dept# che non è conveniente se DEPT è molto piccola

7. Dato che l'attributo Age non è molto selettivo conviene utilizzare un B<sup>+</sup>-Tree.  
Per la group by:

- Indice clustered su Dept#: evita l'ordinamento e la lettura degli impiegati è storizzato in base al dipartimento
- Indice secondario su Dept#: consente di evitare l'ordinamento, ma per ogni dipartimento nelle foglie, devo fare l'accesso alla tabella EMP

```
SELECT Dept#, Count(*)
FROM EMP
WHERE Age>20
GROUP BY Dept#
```

8. Indice unclustered on Dept#, in cui saranno presenti n foglie per ogni Dept. Questo indice si definisce **coprente**, perché risponde alla query in maniera completa.

```
SELECT Dept#, COUNT(*)
FROM EMP
GROUP BY Dept#
```

9. Potrebbe essere conveniente utilizzare un indice composto Age, Salary, esso è il migliore se la condizione su Age è molto selettiva.  
All'interno degli indici composti ordine dei campi è importante, ma risulta difficile da mantenere.

```
SELECT AVG(Salary)
FROM EMP
WHERE Age = 25
AND Salary BETWEEN 3000 AND 5000
```

## 16. Ottimizzazione delle query

L'ottimizzatore seleziona la **strategia migliore** per l'esecuzione della query e garantisce la proprietà di **indipendenza dei dati**, in quanto la forma con cui la query SQL viene scritta non influisce nel modo in cui il risultato viene calcolato, inoltre una riorganizzazione fisica non richiede la riscrittura delle query.

Il piano di esecuzione generato automaticamente dall'ottimizzatore è solitamente il più efficiente:

- Vengono valutate le differenti alternative
- Vengono utilizzate delle statistiche (anche sulla distribuzione dei valori e su come quei valori sono distribuiti nei blocchi fisici) per prendere le decisioni
- Sfrutta le migliori strategie
- Si adatta ai cambiamenti nella distribuzione dei dati

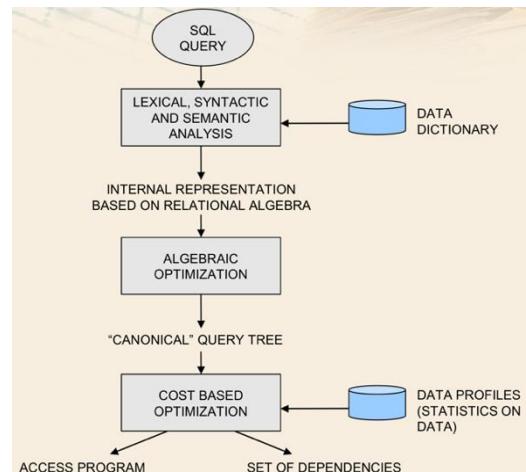
### Operazioni dell'ottimizzatore

L'ottimizzatore esegue principalmente tre operazioni:

- **Analisi lessicale, sintattica e semantica:** la query viene analizzata e vengono individuati
  - Errori lessicali: parole chiave scritte male
  - Errori sintattici: errori relativi alla sintassi SQL
  - Errori semanticci: riferimenti ad oggetti che non esistono nel db, in questo caso viene utilizzato un dizionario contenente i nomi di tutti i vari oggetti presenti nel db

Questa fase restituisce una rappresentazione della query in **algebra relazionale**, questo tipo di rappresentazione mostra l'ordine con il quale gli operatori vengono applicati in maniera **procedurale**.

- **Ottimizzazione algebrica:** esecuzione di trasformazioni algebriche per rendere più efficienti le operazioni (anticipazione della selezione rispetto alla join). L'output è la nuova rappresentazione in **algebra relazionale**.
- **Ottimizzazione sulla base dei costi:** selezione del piano di esecuzione dal punto di vista dei costi di esecuzione:



- Viene selezionato il miglior **metodo di accesso** per ogni tabella
- Viene selezionato il miglior **algoritmo** per ogni operatore relazionale

Infine, viene generato il codice che implementa la migliore strategia.

Questa fase fornisce in output:

- Programma di accesso in formato eseguibile
- Set di dipendenze: condizioni dalle quali dipende la validità del piano di esecuzione (ad esempio l'esistenza di un certo indice)

I **data profiles** vengono utilizzati per la lettura delle statistiche.

### *Modalità di esecuzione delle query*

Ci sono diverse modalità di esecuzione delle query:

- **Compile and go**: dopo la compilazione, il piano di esecuzione viene direttamente eseguito, ma non viene memorizzato (quindi non sarà disponibile in seguito), le dipendenze non sono necessarie.
- **Compile and store**: il piano di esecuzione viene memorizzato assieme alle dipendenze e viene eseguito **on demand**, se la struttura dei dati cambia deve essere ricompilato

### Ottimizzazione algebrica

Nello specifico la fase di **ottimizzazione algebrica** si basa sulle **trasformazioni di equivalenza**, due espressioni relazionali vengono definite **equivalenti** se producono lo stesso risultato.

Queste trasformazioni possono **ridurre la dimensione** dei risultati intermedi per velocizzare le operazioni successive.

Le possibili trasformazioni sono:

1. Atomizzazione della selezione:
2. Proiezioni a cascata:
3. Anticipazione della selezione rispetto alla join (push down della selezione) in cui F è un predicato sugli attributi di E<sub>2</sub>
4. Anticipazione della proiezione rispetto alla join
5. Derivazione della join dal prodotto cartesiano, in cui F contiene solo attributi contenuti in entrambe le tabelle
6. Distribuzione della selezione rispetto all'unione
7. Distribuzione della selezione rispetto alla differenza
8. Distribuzione della proiezione rispetto all'unione
9. Altre proprietà
10. Distribuzione del join rispetto all'unione

$$\sigma_{F_1 \wedge F_2}(E) \equiv \sigma_{F_2}(\sigma_{F_1}(E)) \equiv \sigma_{F_1}(\sigma_{F_2}(E))$$

$$\pi_X(E) \equiv \pi_X(\pi_{X,Y}(E))$$

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie (\sigma_F(E_2))$$

$$\pi_L(E_1 \bowtie_p E_2) \equiv \pi_L((\pi_{L1,J}(E_1)) \bowtie_p (\pi_{L2,J}(E_2)))$$

$$\sigma_F(E_1 \times E_2) \equiv E_1 \bowtie_F E_2$$

$$\sigma_F(E_1 \cup E_2) \equiv (\sigma_F(E_1)) \cup (\sigma_F(E_2))$$

$$\sigma_F(E_1 - E_2) \equiv (\sigma_F(E_1)) - (\sigma_F(E_2)) \equiv (\sigma_F(E_1)) - E_2$$

$$\pi_X(E_1 \cup E_2) \equiv (\pi_X(E_1)) \cup (\pi_X(E_2))$$

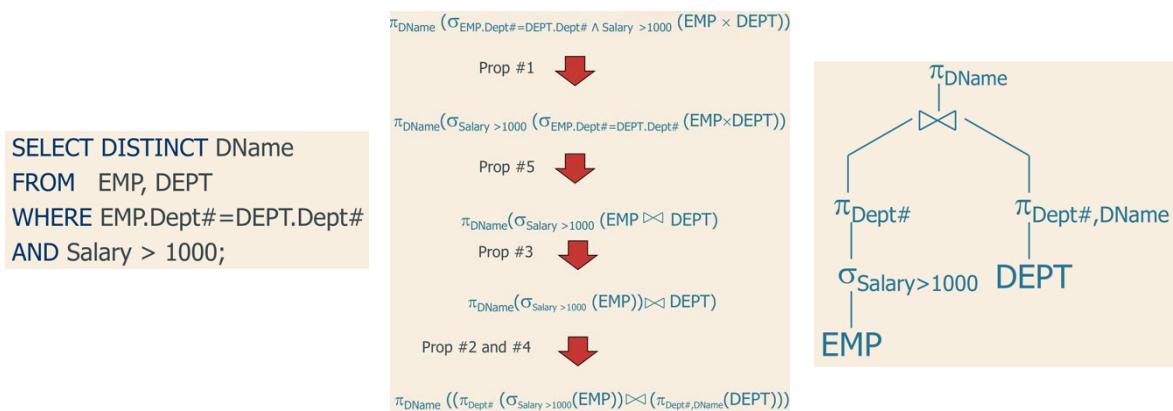
$$\sigma_{F_1 \vee F_2}(E) \equiv (\sigma_{F_1}(E)) \cup (\sigma_{F_2}(E))$$

$$\sigma_{F_1 \wedge F_2}(E) \equiv (\sigma_{F_1}(E)) \cap (\sigma_{F_2}(E))$$

$$E \bowtie (E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E_2)$$

Tutti gli operatori binari sono **commutativi** e **associativi** ad eccezione della differenza

Esempio: Ottimizzazione di una query



### Data profiles

La fase di ottimizzazione basata sui costi utilizza:

- **Data profiles:** informazioni statistiche che descrivono la distribuzione dei dati per le tabelle e per le espressioni relazionali intermedie
- **Formule approssimate per il calcolo dei costi delle operazioni di accesso:** consente la valutazione dei costi delle differenti alternative di esecuzione degli operatori relazionali in un ordine diverso

Nello specifico i **data profiles** sono informazioni quantitative delle caratteristiche delle **tabelle** e delle **colonne**:

- Cardinalità di ogni tabella (e una stima della cardinalità dei risultati intermedi)
- Dimensione in byte delle tuple T
- Dimensione in byte di ogni attributo  $A_j$  in T
- Numero di valori distinti di ogni attributo in T (cardinalità del dominio attivo dell'attributo)
- Valori minimo e massimo per ogni attributo  $A_j$  in T
- Statistiche più fini che caratterizzano la distribuzione vera e propria mediante istogrammi

I profili delle tabelle sono memorizzati nel **data dictionary**, e devono essere periodicamente aggiornati rianalizzando i dati nelle tabelle.

Il comando per l'aggiornamento delle statistiche deve essere inserito dall'utente **on demand**, una esecuzione immediata sovraccaricherebbe il sistema.

I profili sono utilizzati anche per stimare la dimensione delle espressioni relazionali intermedie.

### Query tree

Il **query tree** consiste nella rappresentazione grafica di una espressione relazionale:

- Le **foglie** corrispondono a strutture fisiche come tabelle o indici
- I **nodi intermedi** corrispondono a operazioni sui dati come scan, join o group by

Ci sono diversi tipi di operazioni intermedie:

- **Scan sequenziale:** viene eseguito un accesso sequenziale a tutte le tuple di una tabella (full table scan)
- **Ordinamento:** il cui costo computazionale cresce all'aumentare della dimensione dei dati
- **Valutazione dei predicati di selezione:** è possibile prevedere accessi indicizzati per i predicati selettivi:

- **Uguaglianza** semplice: hash, B<sup>+</sup>-Tree o bitmap
- **Range**: B<sup>+</sup>-Tree
- **Congiunzione di predici (AND)**: l'attributo più selettivo deve essere valutato per primo, rendendo le valutazioni del predicato successivo più snello.  
Una possibile ottimizzazione è quella di calcolare l'intersezione dei bitmap/RID provenienti dagli indici per poi leggere dalla tabella solo le tuple individuate.
- **Disgiunzione di predici (OR)**: è possibile sfruttare gli indici solo se tutti i predici (tutti i loro attributi) hanno un indice, altrimenti va fatto un full table scan

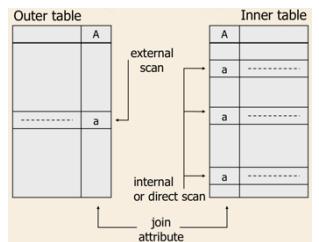
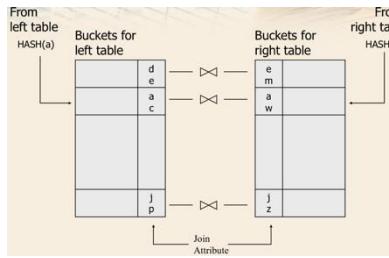
- **Join**: esistono diversi algoritmi per effettuare il join:

- **Nested loop**: dopo aver individuato una **inner** table (la più piccola) e una **outer** table, per ogni tupla dell'outer table viene effettuata una scansione completa della inner table.

Quindi viene fatto un singolo full scan della outer table, e per ogni sua tupla viene fatto un full scan della inner table.

L'accesso alla inner table può essere **velocizzato** in due modi:

- Viene caricata in memoria
- Viene utilizzato un indice sull'attributo di join



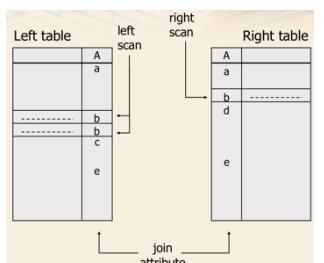
È ottimale quando la inner table è **ordinata** e abbastanza **piccola** da entrare in memoria (in assenza di indice).

Viene utilizzato quando le due tabelle sono molto **sbilanciate**.

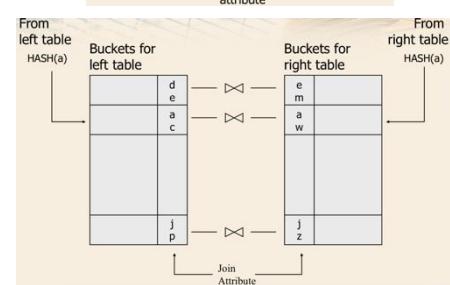
Questo algoritmo è **non simmetrico**, infatti scambiando le tabelle il costo di esecuzione aumenta.

- **Merge scan**: richiede che le due tabelle siano già **ordinate** (rispetto all'attributo di join) e consiste in uno scan parallelo che di volta in volta genera le coppie di tuple che soddisfano la condizione di join.

Il costo di esecuzione è **simmetrico**.



- **Hash join**: le tuple di entrambe le tabelle vengono bucketizzate rispetto all'attributo di join in modo che possa essere possibile effettuare un veloce ordinamento interno al bucket per poi effettuare la join fra le tuple all'interno degli stessi bucket delle due tabelle  
È conveniente per tabelle **molto grandi**.



- **Bitmapped join**: è un indice precompilato che consiste in una matrice di bit in cui le righe e le colonne contengono i RID delle due tabelle. Una cella contiene 1 se la riga *i* soddisfa la condizione di join con la colonna *j*.

L'aggiornamento potrebbe risultare **lento**.

È utilizzato nei sistemi di **DW** per la join fra la tabella dei fatti e le dimensioni.

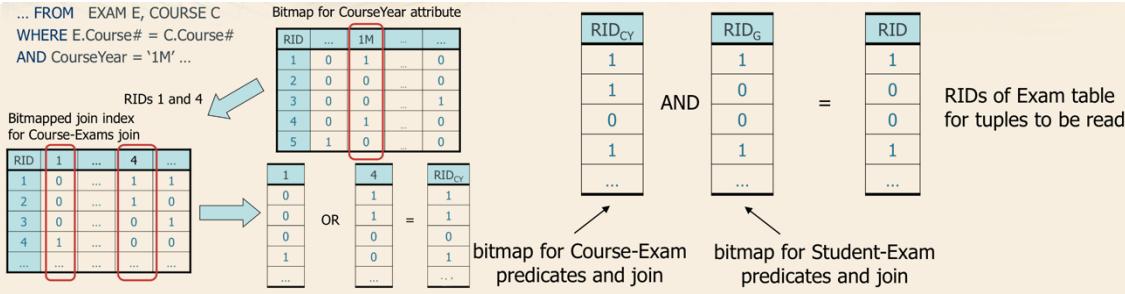
| RID | 1   | 2   | ... | n |
|-----|-----|-----|-----|---|
| 1   | 0   | 0   | ... | 1 |
| 2   | 0   | 1   | ... | 0 |
| 3   | 0   | 0   | ... | 1 |
| 4   | 1   | 0   | ... | 0 |
| ... | ... | ... | ... | 0 |

Può sfruttare l'utilizzo di **indici bitmapped** definiti sull'attributo di join per entrambe le tabelle.

Esempio:

- STUDENT (Reg#, SName, Gender)
- COURSE (Course#, CName, CourseYear)
- EXAM (Reg#, Course#, Date, Grade)

```
SELECT AVG (Grade)
FROM STUDENT S, EXAM E, COURSE C
WHERE E.Reg# = S.Reg#
AND E.Course# = C.Course#
AND CourseYear = '1M'
AND Gender = 'M';
```



Mediante l'indice bitmapped su CourseYear vengono selezionati i RID corrispondenti al primo anno magistrale. In seguito, isolando nella matrice del bitmapped join, le colonne corrispondenti ai RID e mettendole in OR logico, otteniamo la lista di RID che soddisfano la condizione di join. Infine, mediante una AND fra i RID che corrispondono ai due predicati di selezione otteniamo la lista di RID di Exam che deve essere considerata per il calcolo della media. In questo modo l'accesso in memoria sarà più veloce.

- Group by:
  - **Sort based**: le tuple vengono ordinate in base all'attributo di join e poi vengono calcolati gli aggregati dei gruppi
  - **Hash based**: viene applicata una funzione di hashing per bucketizzare le tuple che in seguito vengono ordinate e vengono calcolati gli aggregati
  - **Viste materializzate**: se la funzione di query rewrite è attiva sulla vista, l'ottimizzatore può riscrivere le query sql in modo da sfruttare il raggruppamento preventivo effettuato da essa

Nel caso in cui il sort o l'hashing siano stati già effettuati posso eseguire una group by **no-sort** o **no-hash**.

### Individuazione del piano di esecuzione

L'individuazione del piano di esecuzione migliore da parte dell'ottimizzatore dipende da diversi fattori:

- Il modo in cui i dati vengono letti dalla memoria
- L'ordine di esecuzione dei diversi operatori
- La tecnica di implementazione dei singoli operatori
- Quando e se deve essere effettuato un ordinamento

In generale l'ottimizzatore individua un **albero contenente le diverse alternative** in cui ogni nodo interno rappresenta una decisione su uno dei fattori e le foglie rappresentano i diversi piani di esecuzione individuati.

Esempio: considerando 3 tabelle su cui applicare la join ( $T_1 \text{ join } T_2 \text{ join } T_3$ ), abbiamo 4 diverse tecniche di join (per entrambe le join) e 3 diversi ordini.

Otterremo in totale un albero con  $4 * 4 * 3 = 48$  nodi foglia (possibili piani di esecuzione).

L'ottimizzatore andrà a selezionare il nodo foglia con un costo minore, il costo viene calcolato come  $C_{Total} = C_{I/O} * n_{I/O} + C_{cpu} * n_{cpu}$  in cui abbiamo la somma fra il costo complessivo per le operazioni di I/O e quello per le operazioni CPU.

## L'ottimizzatore di Oracle

L'ottimizzatore, in Oracle, svolge diverse funzioni:

- Valutazione di espressioni e condizioni
- Riscrive l'operazione SQL (ad esempio query rewrite)
- Sceglie l'obiettivo dell'ottimizzatore (a seconda dell'obiettivo il piando di esecuzione scelto cambia)
- Sceglie i percorsi di accesso ai dati (se utilizzare o meno certi indici) in base agli indici disponibili
- Sceglie l'ordine delle join
- Sceglie l'algoritmo di esecuzione della join

Il recupero dei dati dal database può avvenire in diversi modi:

- **Full table scans:** questo tipo di scan consente di leggere tutte le tuple di una tabella eliminando coloro che non rispettano i criteri di selezione (where).  
I blocchi da leggere sono tutti adiacenti e vengono letti in maniera sequenziale, è possibile inoltre effettuare delle letture **multiblocco** per leggere più blocchi in una singola chiamata I/O.

Le decisioni dell'ottimizzatore sono influenzate dall'**index clustering factor** che indica quanto sono sparsi i dati relativi a valori di colonne simili all'interno dei blocchi della tabella:

- Se è **basso** le righe con valore dell'attributo simile saranno concentrate negli stessi blocchi, in questo caso la creazione di un indice è conveniente
- Se è **alto** le righe con valore dell'attributo simile sono distribuite fra i vari blocchi, in questo caso la creazione di un indice non è conveniente (l'indice farà riferimento, comunque, a tutti i blocchi e un full scan potrebbe essere più veloce)
- **Index scans:** su Oracle è possibile definire diversi tipi di indici primari (clustered Btree, Hash) e secondari (Btree, Bitmap, Hash) ma non è possibile stabilire come devono essere fisicamente realizzati.

Gli indici contengono il **valore indicizzato** e i **rowid** delle righe nella tabella che hanno quel valore, esistono diversi tipi di index scan:

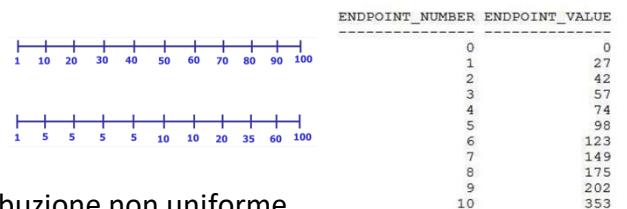
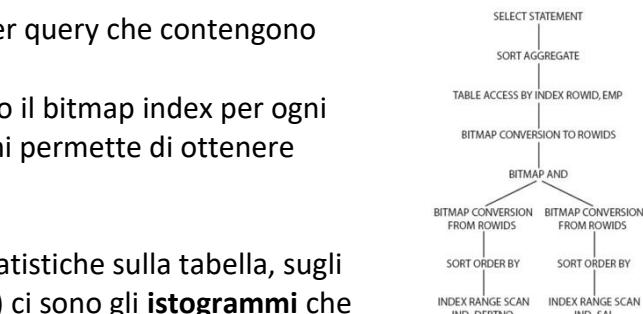
- **Index Unique Scans:** questo scan restituisce al massimo **un** ROWID per ogni valore indicizzato (l'attributo indicizzato è **unique**)
- **Index Range Scans:** dalla lettura dell'indice si ottengono **diversi** ROWID **ordinati** in maniera ascendente.  
Può essere utile quando le operazioni successive richiedono ordinamento rispetto all'attributo indicizzato.
- **Index Full Scans:** è disponibile quando il predicato contiene una delle colonne nell'indice, o quando non c'è un predicato.  
Restituisce i dati dell'indice ordinati in base ai valori memorizzati nell'indice.

- **Fast Full Index Scans:** è un'alternativa al full table scan utilizzata quando l'indice è coprente, consiste nell'utilizzare l'indice senza accedere alla tabella. L'indice viene letto mediante lettura **multiblocco** quindi il risultato non è ordinato.
- **Rowid scans (access by rowid):** consente di accedere alla tabella mediante il ROWID per prelevare il blocco contenente la tupla desiderata. L'indirizzo fisico del blocco contenente la tupla viene calcolato a partire dal ROWID.  
È il metodo più veloce per recuperare una **singola** tupla.
- **Bitmap index:** gli indici di tipo bitmap sono efficaci per query che contengono condizioni multiple nella where.  
Nel contesto del piano di esecuzione viene prima letto il bitmap index per ogni attributo e in seguito viene effettuata una AND che mi permette di ottenere l'insieme di ROWID che soddisfano le condizioni.

Fra le diverse **statistiche** a disposizione dell'ottimizzatore (statistiche sulla tabella, sugli attributi, sugli indici e sulle performance sistema in generale) ci sono gli **istogrammi** che raccolgono stime accurate sulla distribuzione dei dati relativamente ad un attributo.

Gli istogrammi possono essere di due tipi:

- **Height-balanced histograms:** i possibili valori dell'attributo vengono divisi in fasce che contengono lo stesso numero di record.  
Nella figura a lato, l'istogramma in alto rappresenta una distribuzione uniforme, quello in basso una distribuzione non uniforme.  
Nella rappresentazione Oracle la prima colonna rappresenta la fascia considerata, e la seconda il valore massimo all'interno di essa.
- **Frequency histograms:** per ogni possibile valore ne viene specificato il numero di occorrenze.  
Nella rappresentazione Oracle la prima colonna rappresenta la frequenza dei valori e la seconda il valore considerato.



| ENDPOINT_NUMBER | ENDPOINT_VALUE |
|-----------------|----------------|
| 36              | 1              |
| 213             | 2              |
| 261             | 3              |
| 370             | 4              |
| 484             | 5              |
| 692             | 6              |
| 798             | 7              |
| 984             | 8              |
| 1112            | 9              |

È possibile scegliere in che modalità l'ottimizzatore deve lavorare:

- Ottimizzazione per il **throughput** migliore: l'ottimizzatore cerca di utilizzare il minor numero di risorse possibile
- Ottimizzazione per i **tempi di risposta** migliori: l'ottimizzatore cerca di restituire i valori delle prime righe della query nel minor tempo possibile.

## Hint in Oracle

Gli **hint** rappresentano un meccanismo per fornire istruzioni sulla scelta di un certo piano di esecuzione da parte dell'ottimizzatore.

Esistono diversi tipi di hint:

- Obiettivo e approccio di ottimizzazione:
  - **ALL\_ROWS:** l'obiettivo è l'ottenimento di un throughput ottimale (consumo minimo delle risorse)
  - **FIRST\_ROWS(n):** l'obiettivo è una risposta veloce, scegliendo il piano di esecuzione che produca le prime n righe in maniera più efficiente

- Modalità di accesso:
  - **FULL(table)**: viene utilizzato il full table scan sulla tabella specificata
  - **INDEX(table indexName1 ...)**: viene utilizzato index scan mediante uno degli indici specificati
  - **NO\_INDEX(table indexName1 ...)**: viene evitato l'utilizzo degli indici specificati
  - **INDEX\_COMBINE(table indexName1 ...)**: viene utilizzata la modalità di accesso bitmap per gli indici specificati
  - **INDEX\_FFS(table indexName1 ...)**: dice all'ottimizzatore di utilizzare un fast full index scan piuttosto che un full table scan sugli indici specificati
  - **NO\_INDEX\_FFS(table indexName1 ...)**: esclude l'utilizzo di un fast full index scan per gli indici specificati
- Trasformazione delle query
- Ordine di join:
  - **ORDERED**: il join viene fatto seguendo l'ordine specificato nella FROM
  - **LEADING(table1,table2,...)**: l'ottimizzatore utilizza l'ordine specificato  
In caso di NL la tabella specificata per prima viene utilizzata come outer table.
- Operazione di join utilizzata: **[NO\_]USE\_NL(table1,...)**, **[NO\_]USE\_MERGE**, **[NO\_]USE\_HASH**
- Esecuzione parallela
- Altri hint

## 17. Gestione della concorrenza

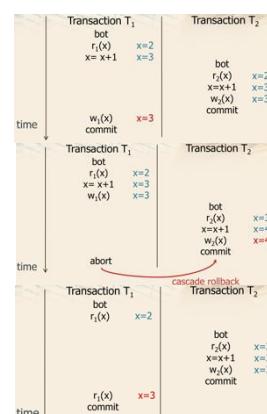
Il controllo di concorrenza fornisce la possibilità di accesso concorrente ai dati incrementando l'efficienza del DBMS che diventa in grado di **massimizzare il numero di transazioni** eseguite al secondo, e **minimizzare il tempo di risposta** medio alle transazioni.

Le operazioni di I/O elementari che devono essere gestite sono quelle di **lettura e scrittura** che possono richiedere l'accesso ad un'intera pagina.

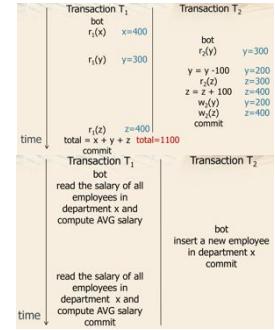
### Anomalie

Lo **scheduler** è il componente che gestisce il controllo di concorrenza e decide quando le richieste di lettura e scrittura devono essere soddisfatte, l'assenza di tale componente può causare delle **anomalie** che possono essere di diverso tipo:

- **Lost update**: l'aggiornamento fatto da T2 su x, viene perso perché sovrascritto da quello fatto da T1 che ha letto x prima della scrittura di T2. Il valore corretto di x dovrebbe essere 4.
- **Dirty read**: T2 legge il valore di x in uno stadio intermedio che non diventa permanente, in quanto T1 abortisce.
- **Inconsistent read**: la transazione T1 legge x due volte, ma i due valori letti sono differenti



- **Ghost update (a):** La transazione T1 osserva solo parzialmente gli effetti della transazione T2; infatti, legge x e y prima dell'aggiornamento di T2 e dopo la fine di T2 legge z. È stato quindi perso l'aggiornamento su y fatto da T2. Il risultato sarebbe dovuto essere 1000.
- **Ghost update (b):** in questo caso le due medie calcolate sono diverse e ciò è dovuto all'inserimento che rappresenta l'aggiornamento "fantasma". La differenza rispetto all'inconsistent read è che in questo caso l'oggetto è stato inserito proprio durante T1. Non è possibile fare lock su un oggetto che ancora non esiste ma che è stato appena inserito.



## Schedule

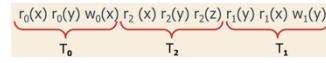
Dal punto di vista teorico, nel controllo della concorrenza ci sono due definizioni importanti:

- **Transazione:** una sequenza di operazioni di lettura e scrittura che hanno lo stesso TID
- **Schedule:** una sequenza di operazioni di lettura e scrittura appartenenti a transazioni concorrenti. Le operazioni appaiono nell'ordine di arrivo.

Lo scheduler durante il controllo di concorrenza accetta o rifiuta certe schedule per evitare la comparsa di anomalie, e fa ciò senza conoscere l'esito finale della transazione (commit/abort).

Lo scheduler applica quindi la **commit projection**, un'ipotesi semplificativa che afferma che ogni transazione farà **commit**. L'anomalia di **dirty read** non viene quindi gestita.

Una **schedule seriale** è una schedule in cui le azioni di ogni transazione appaiono in sequenza, senza operazioni interlacciate che appartengono a transazioni differenti.



Una **schedule  $S_i$**  arbitraria si definisce **corretta/serializzabile** se fornisce gli stessi risultati di una qualsiasi schedule **seriale  $S_j$** .

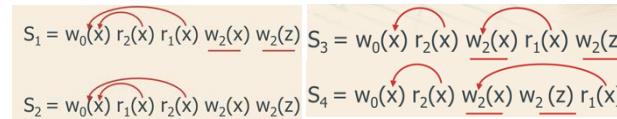
## Classi di equivalenza

Esistono diverse **classi di equivalenza** fra le schedule, ognuna delle quali rileva un set di schedule accettabili e che hanno complessità diversa nell'individuazione dell'equivalenza:

- **Timestamp equivalence**
- **View equivalence:** due schedule sono view equivalent se hanno gli stessi:
  - **reads-from:**  $r_i(x)$  reads-from  $w_j(x)$  se  $w_j(x)$  precede  $r_i(x)$  con  $i \neq j$  e non c'è nessuna  $w_k(x)$  fra di esse
  - **final write:**  $w_i(x)$  è una final write se essa è l'ultima scrittura di x nella schedule

Una schedule è **view serializzabile** se è view equivalente ad una schedule seriale arbitraria della stessa transazione.

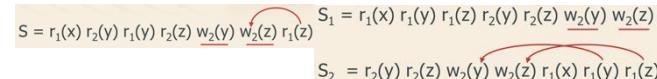
$S_1$  è view serializzabile perché è view equivalent a  $S_2$   
 $S_3$  non è view equivalente a  $S_2$ , ma lo è con  $S_4$ , quindi è view serializzabile



Analizziamo come si comporta rispetto alle varie anomalie:

- **Lost update:** non è view serializzabile in quanto non è view equivalente a nessuno dei due possibili schedule seriali. Viene quindi rigettato.
- **Inconsistent read:** non è view serializzabile per lo stesso motivo e viene rigettato.

- **Ghost update(a)**: anche in questo caso S non è view serializzabile

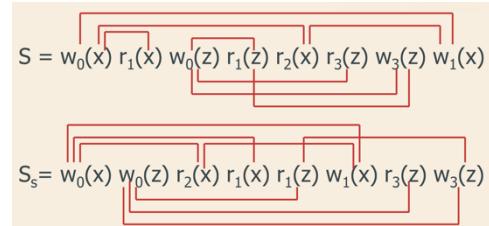


La complessità di determinare la view equivalence ad una schedule seriale arbitraria è lineare, mentre la determinazione della view equivalence con un qualsiasi schedule seriale arbitrario non è realizzabile nei sistemi reali.

Per questo motivo, per le applicazioni reali, si sono scelte tecniche più veloci ma meno accurate.

- **Conflict equivalence**: due schedule sono conflict equivalent se presentano gli stessi conflitti e ogni coppia di azioni è nello stesso ordine in entrambe le schedule.

Due azioni  $A_i$  e  $A_j$  sono in conflitto se operano sullo stesso oggetto e una delle due azioni è una scrittura (RW, WR o WW). Una schedule è **conflict serializzabile** (CSR) se è equivalente ad una schedule seriale arbitraria delle stesse transazioni.

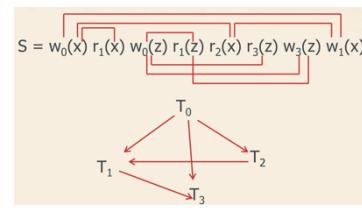


Per individuare la conflict serializability è possibile sfruttare un **grafo dei conflitti** in cui ogni nodo rappresenta una transazione e ogni arco rappresenta il fatto che c'è almeno un conflitto fra  $A_i$  di  $T_i$  e  $A_j$  di  $T_j$  e  $A_i$  precede  $A_j$ .

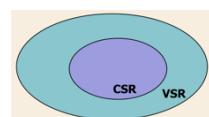
Se il grafo dei conflitti è aciclico lo schedule è CSR.

Questa classe di equivalenza, mediante l'utilizzo del grafo, ha un complessità **lineare** rispetto alla dimensione del grafo.

Nell'esempio S è conflict serializzabile perché il grafo ottenuto è aciclico.



Nei sistemi reali anche questo approccio non è l'ideale, in quanto considerando  $100\text{tps} * 5\text{s} = 500$  nodi \* 10 pagine per transazione avremmo circa 5000 accessi a memoria con conseguenti modifiche al grafo e controllo di aciclicità.



All'effettivo le schedule CSR rappresentano un sottoinsieme di quelle VSR.

- **2 phase locking**: un **Lock** è un blocco su una risorsa che ne previene l'utilizzo da parte di altri, le operazioni di lock possono essere di read (R-Lock) e di write (W-Lock), dopo l'utilizzo della risorsa lo sblocco di essa avviene mediante **Unlock**.

La R-Lock è **condivisa** fra le transazioni (diverse transazioni possono eseguire la R-Lock sulla stessa risorsa che vengono contate mediante un counter) mentre la W-Lock è **esclusiva** e non è compatibile con altre operazioni di R/W-Lock sugli stessi dati.

La **lock escalation** è la richiesta di una R-Lock seguita da una W-Lock sugli stessi dati.

Lo scheduler diventa un **lock manager** ricevendo le richieste da parte delle transazioni e concedendo i lock in base ai lock già concessi alle altre transazioni.

Quando una richiesta di lock viene **concessa**, la corrispondente risorsa viene acquisita dalla transazione richiedente, che, quando ha finito, effettua l'Unlock rendendo la risorsa di nuovo disponibile.

Quando la lock **non** viene concessa la transazione viene messa in uno stato di attesa che finisce quando la risorsa viene sbloccata tornando disponibile.

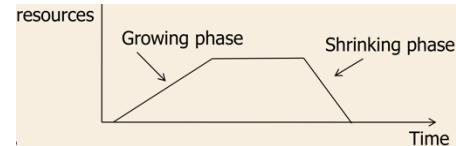
Per eseguire queste operazioni il lock manager sfrutta:

- **Lock table:** tabella memorizzata nella memoria principale che consente di decidere se una lock può essere concessa o meno. Per ogni oggetto di dati vengono utilizzati 2 bit per rappresentare i 3 possibili stati (free, r\_locked, w\_locked) e un contatore che conti il numero di transazioni in attesa della risorsa
- **Conflict table:** per gestire i conflitti fra le transazioni

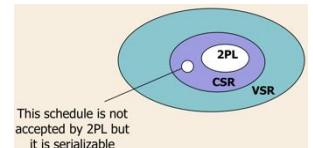
| Request | Resource State |                                           |             |
|---------|----------------|-------------------------------------------|-------------|
|         | Free           | R-Locked                                  | W-Locked    |
| R-Lock  | Ok/R-Locked    | Ok/R-Locked                               | No/W-Locked |
| W-Lock  | Ok/W-Locked    | No/R-Locked                               | No/W-Locked |
| Unlock  | Error          | Ok/It depends (free if no other R-Locked) | Ok/Free     |

Il 2 Phase Locking viene utilizzato dalla maggior dei DBMS commerciali ed è caratterizzato da due fasi:

- **Growing phase:** durante questa fase la transazione può richiedere i lock
  - **Shrinking phase:** durante questa fase devono essere rilasciati tutti i lock e non è possibile richiederne dei nuovi
- Fra le due fasi la transazione esegue le proprie operazioni sui dati.

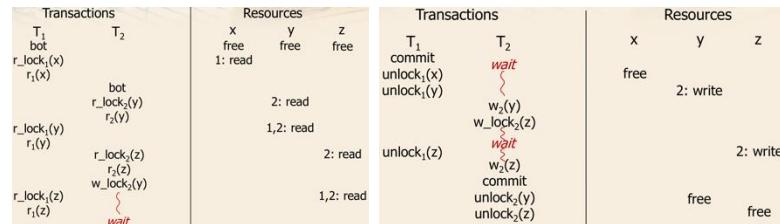


Il 2 Phase Locking garantisce serializzabilità e rappresenta un sottoinsieme delle transazioni CSR.



Dall'esempio si nota che la transazione analizzata è CSR in quanto il grafico è aciclico ma non 2PL in quanto già dopo la prima striscia nera si entra in fase di shrinking ma alla seconda striscia nera viene richiesto il lock di un'altra risorsa.

Guardando l'esempio successivo invece si nota che il 2PL riesce a gestire correttamente il Ghost update (a).



Lo **strict 2 phase locking** è una restrizione che prevede che una transazione deve rilasciare le risorse solo dopo il commit/abort e quindi alla fine della transazione, in modo tale che i dati siano stabili ed evitare l'anomalia di dirty read.

Le primitive esistenti sono:

- **R-Lock** (T, x, ErrorCode, TimeOut)
- **W-Lock** (T, x, ErrorCode, TimeOut)
- **UnLock** (T, x)

Con:

- **T:** TID
- **x:** risorsa richiesta
- **ErrorCode:** parametro di ritorno che può essere **Ok** o **Not Ok**
- **TimeOut:** tempo di attesa massimo della transazione

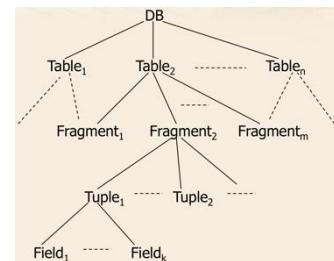
Quando una transazione richiede una risorsa x:

- Se la richiesta può essere soddisfatta: il lock manager modifica lo stato di x nella sua tabella e restituisce il codice **Ok**. In questo caso l'attesa è piccola perché la risorsa è libera.
- Se la richiesta **non** può essere soddisfatta al momento: la transazione richiedente viene inserita in una coda di attesa, quando la risorsa torna disponibile la prima transazione in coda ottiene il lock della risorsa e riprende l'esecuzione
- Quando il **timeout scade** mentre la transazione è in attesa, il lock manager estrae la transazione dalla coda, la fa ripartire e restituisce il codice **Not Ok**. La transazione potrebbe eseguire un rollback per poi ricominciare, oppure, dopo un po' di tempo, richiedere lo stesso lock ancora una volta.

La probabilità di avere conflitti viene calcolata come  $(K * M)/N$ :

- K: numero di transazioni attive
- M: numero medio di oggetti a cui ogni transazione fa accesso
- N: numero di oggetti nel db

## Locking gerarchico



Mediante il locking gerarchico possono essere acquisiti lock a granularità differente: tabelle, gruppi di tuple, singole tuple, singoli campi in una tupla.

Questo tipo di locking estende quello tradizionale ed è caratterizzato da alcune primitive aggiuntive:

- **Shared Lock (SL)**: effettua una lettura
- **eXclusive Lock (XL)**: effettua una scrittura
- **Intention of Shared Lock (ISL)**: mostra l'intenzione di eseguire uno shared lock su uno oggetto presente in un nodo figlio
- **Intention of eXclusive Lock (IXL)**: analogo a ISL, ma per l'eXclusive lock
- **Shared lock and Intention of eXclusive Lock (SIXL)**: effettua uno shared lock dell'oggetto corrente e mostra l'intenzione di eseguire un eXclusive lock per uno o più oggetti all'interno dei nodi figli

Il **request protocol** descrive come si comporta il lock manager per gestire questo tipo di scenari:

1. I lock vengono sempre concessi a partire dal nodo radice e scendendo fino ai nodi foglia
2. I lock vengono rilasciati a partire dal nodo con un lock a granularità più dettagliata a salire
3. Per richiedere un SL o un ISL di un dato nodo, una transazione deve avere un ISL o un IXL sul suo nodo superiore
4. Per richiedere un XL, IXL o SIXL su un nodo, una transazione deve avere un IXL o un SIXL sul suo nodo superiore

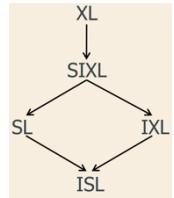
Nella **matrice di compatibilità** vengono mostrate tutte le possibili combinazioni:

- In questo caso i due lock non sono compatibili in quanto XL è esclusivo e non è possibile accedere in lettura ad un figlio
- In questo caso SL non viene concesso perché qualcuno ha bloccato il figlio in scrittura
- In questo caso i due lock sono compatibili perché abbiamo due accessi in lettura

|         | Resource State |     |    |      |    |
|---------|----------------|-----|----|------|----|
| Request | ISL            | IXL | SL | SIXL | XL |
| ISL     | Ok             | Ok  | Ok | Ok   | No |
| IXL     | Ok             | Ok  | No | No   | No |
| SL      | Ok             | No  | Ok | No   | No |
| SIXL    | Ok             | No  | No | No   | No |
| XL      | No             | No  | No | No   | No |

- In questo caso IXL non viene concesso perché la risorsa padre è bloccata in lettura e la transazione corrente vuole modificarne una parte.

Di fianco è rappresentato il grafo delle precedenze per i lock.



La scelta della granularità dipende dal tipo di applicazione:

- Se esegue letture e update **localizzati** di pochi oggetti allora si utilizza una granularità più dettagliata. Ciò riduce la concorrenza e rende i conflitti più probabili.
- Se esegue letture e update **massivi** allora si utilizza una granularità più grezza. Ciò comporta un overhead del lock manager.

### *Locking dei predici*

Il **locking dei predici** consente di indirizzare l'anomalia di **Ghost Update(b)**, di base in 2PL un'operazione di lettura non è in conflitto con l'inserimento di una tupla perché essa non può essere bloccata in anticipo.

Il locking del predico invece permette di bloccare tutti i dati che soddisfano un certo predicato. Questo tipo di locking viene implementato nei sistemi reali bloccando gli indici.

### *Livelli di isolamento*

Nello standard **SQL2** le transazioni possono essere di tipo **read-write** o di tipo **read-only**.

Il **livello di isolamento** di una transazione specifica come essa interagisce con le altre transazioni in esecuzione e viene settato mediante comando SQL.

I livelli di isolamento sono:

- SERIALIZABLE**: è il livello più alto di isolamento, include il **locking dei predici** e blocca tutte le anomalie
- REPEATABLE READ: strict 2PL** senza locking dei predici, non protegge dalla Ghost Update(b)
- READ COMMITTED**: nessun 2PL, il lock di lettura viene rilasciato non appena l'oggetto è stato letto, viene evitata la lettura di stati intermedi evitando quindi **dirty read**
- READ UNCOMMITTED**: nessun 2PL, i dati vengono letti **senza lock** permettendo letture sporche, è consentito solo per transazioni **read-only**

In SQL2 il livello di isolamento può essere settato mediante:

**SET TRANSACTION [ISOLATION LEVEL <IsolationLevel>] [READ ONLY] [READ WRITE]**

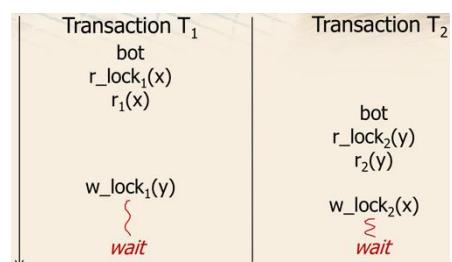
Il livello di isolamento può essere **ridotto solo per operazioni di lettura**, le operazioni di write vengono sempre eseguite almeno in **strict 2PL con lock esclusivo**.

### *Deadlock*

Il **deadlock** è una situazione tipica che capita in sistemi ricorrenti che lascia due transazioni bloccate in attesa che l'altra faccia l'Unlock sulla risorsa richiesta.

La soluzione più comunemente utilizzata ai deadlock è l'utilizzo di un **timeout** passato il quale la transazione in attesa riceverà una risposta negativa ed effettuerà il rollback.

Un timeout troppo **lungo** genera dei deadlock lunghi, un timeout troppo **corto** va a sovraccaricare il sistema.



Altre possibili soluzioni sono:

- **Pessimistic 2PL**: tutti i lock vengono acquisiti prima dell'inizio della transazione, non è realizzabile perché andrebbe a rallentare troppo il sistema
- **Timestamp**: solo le transazioni più "giovani" (o più "vecchie") possono rimanere in attesa che la transazione più "vecchia" (o più "giovane") venga abortita.

La rilevazione dei deadlock è basata sul **wait graph** in cui ogni freccia rappresenta un stato di attesa fra due transazioni, un ciclo nel grafo rappresenta un deadlock.

Un grafo di questo tipo è complicato da realizzare e mantenere, viene utilizzato solitamente all'interno di **DBMS distribuiti**.

## 18. Reliability manager

Il **reliability manager** si occupa di attuare e far rispettare le proprietà ACID di **atomicità** e **durabilità**.

Utilizza un file di **log**, che contiene tutte le attività svolte dal DBMS, memorizzato all'interno di una memoria **stabile** (con una probabilità di failure tendente a 0 e scorrelata dalle altre memorie utilizzate).

Le operazioni vengono memorizzate sequenzialmente in ordine cronologico e possono essere di due tipi:

- Record transazionali:

- **Begin**            B(T)
- **Commit**        C(T)
- **Abort**          A(T)
- **Insert**         I(T, O, AS)
- **Delete**        D(T, O, BS)
- **Update**       U(T, O, BS, AS)

In cui:

- **T** è l'id della transazione
- **O** è l'id del record interessato (RID)
- **AS** è l'After State
- **BS** è il Before State

In caso di **undo** di un'azione relativa ad un oggetto:

- **Insert O**: bisogna eliminare l'oggetto O
- **Update O**: bisogna scrivere il BS di O
- **Delete O**: bisogna scrivere il BS di O

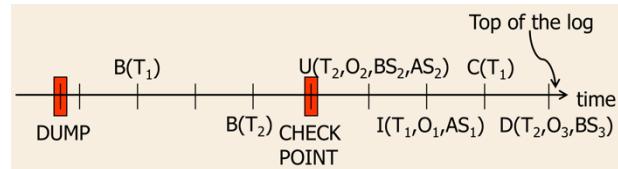
In caso di **redo** di un'azione relativa ad un oggetto:

- **Insert O**: bisogna scrivere il AS di O
- **Update O**: bisogna scrivere il AS di O
- **Delete O**: bisogna eliminare O

Queste operazioni sono utili in caso di crash e devono essere idempotenti.

- Record di sistema:

- **Checkpoint CK( $T_1, T_2, \dots, T_n$ )**: specifica l'insieme delle transazioni attive al momento.  
È un'operazione richiesta periodicamente dal Reliability Manager al Buffer Manager che consente un processo di recupero più veloce.



Tiene traccia dei TID di tutte le transazioni attive e memorizza in maniera sincrona (primitiva force) su disco le pagine relative alle transazioni conclusive. Durante queste operazioni nessuna transazione può fare commit.

Al termine il record di checkpoint viene scritto in maniera sincrona sul file di log.

- **Dump**: crea una copia completa della base dati memorizzandola in una memoria stabile.  
Questa operazione viene solitamente eseguita quando i sistemi sono offline.

Alla fine, viene scritto un record di dump nel file di log, assieme al timestamp e al dispositivo di dump utilizzato.

Ci sono due regole principali per la scrittura del log:

- **Write Ahead Log (WAL)**: il BS dei dati in un record di log deve essere scritto prima che il dato venga scritto nel database. Ciò consente operazioni di undo su dati parzialmente scritti nel disco.
- **Commit precedence**: il AS dei dati in un record di log deve essere scritto prima dell'esecuzione dell'operazione di commit. Ciò consente operazioni di redo per transazioni che hanno già eseguito commit, ma che non sono state scritte nel database.

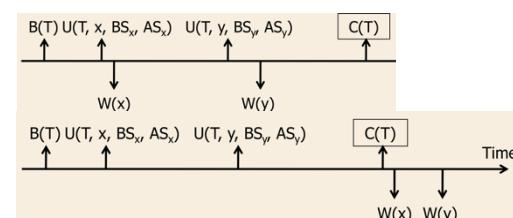
All'effettivo BS e AS vengono scritti insieme in quanto decidiamo di scrivere il log prima del record nel database e prima del commit.

Inoltre, il log viene scritto in maniera **sincrona** per i commit e per la scrittura dei dati su disco, in maniera **asincrona** per abort/rollback.

Il **record di commit** relativo ad una transazione rappresenta il limite prima del quale la transazione deve essere **sfatta** in caso di failure, dopo il quale la transazione deve essere **rieseguita** in caso di failure.

Esistono diversi **protocolli** per la scrittura del log e del database:

- Tutte le scritture del database su disco vengono eseguite **prima del commit**: non richiede il **redo** delle transazioni committate
- Tutte le scritture vengono eseguite **dopo il commit**: non richiede **undo** di transazioni non committate



- Le scritture possono avvenire **sia prima che dopo il commit**: richiede sia operazioni di undo che operazioni di redo

Nei sistemi reali viene utilizzato un approccio misto.

L'utilizzo di un protocollo robusto durante la scrittura del file di log per assicurarne l'affidabilità è **costoso** (il costo è comparabile a quello di un update del db).

Questo protocollo deve essere in grado di garantire le proprietà ACID e ottimizza la scrittura del log grazie a:

- Una forma **compatta**
- Parallelismo
- Commit di **gruppi** di transazioni

## Gestione del ripristino

Esistono due tipi di failures:

- System failure**: guasto causato da problemi sw o dall'interruzione dell'alimentazione. Generano la **perdita della memoria principale** ma non del disco.
- Media failure**: questi guasti sono causati da un guasto del dispositivo che gestisce la memoria secondaria. Generano la **perdita del contenuto su disco**, ma non del file di log.

Quando si verifica un guasto il sistema si **ferma**.

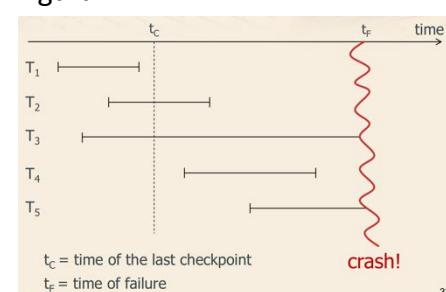
Quando il sistema riparte entra in fase di recovery, che può essere di due tipi:

- Warm restart**: utilizzato in caso di system failure

Immaginiamo che il sistema si trovi nella situazione rappresentata in figura.

Notiamo che al momento del guasto:

- L'unica transazione **completata** prima del checkpoint è **T1** (non è richiesta alcuna operazione di recupero).
- Le transazioni **committate** (ma di cui non ne è certa la scrittura) sono **T2** e **T4** (è necessaria una redo).
- Le transazioni attive sono **T3** e **T5** (dato che non hanno committato è necessaria una undo).



Per la procedura di recovery non è necessaria la presenza di un checkpoint, come si nota esso serve solo per renderla più veloce. Se non ci fosse il record di checkpoint si dovrebbe rileggere l'intero log fino all'ultimo dump.

Algoritmo di warm restart:

- Lettura all'indietro del log fino all'ultimo checkpoint
- Individuazione delle transazioni sui cui eseguire undo/redo
  - Al checkpoint creazione delle liste:
    - UNDO = {Transazioni attive al checkpoint}
    - REDO = {Vuota}
  - Lettura in avanti del log
    - In UNDO vengono aggiunte le transazioni per cui viene trovato il **begin** record

- In REDO vengono spostate da UNDO le transazioni per cui viene trovato il **commit** record
  - Le transazioni che hanno fatto ABORT restano in UNDO
- Alla fine di questa fase abbiamo le due liste delle transazioni su cui fare undo/redo.
- Ripristino dei dati
    - Il log viene letto all'indietro disfacendo le operazioni delle transazioni nella lista di UNDO fino al **begin** record della transazione più vecchia (anche se è precedente al checkpoint) nella lista di UNDO.
    - Il log viene riletto in avanti e le operazioni svolte dalle transazioni nella lista REDO vengono applicate al database.
  - **Cold restart:** utilizzato quando viene danneggiato il database su disco.  
I passi principali sono:
    - Accesso all'ultimo dump per ripristinare la porzione di memoria danneggiata
    - Partendo dall'ultimo record di dump, il log viene letto in avanti e vengono rieseguite tutte le operazioni sul database, assieme alle operazioni di commit/rollback
    - Viene eseguito un warm restart per rivedere le operazioni successive all'ultimo checkpoint, eseguite dal dump facendo undo e redo quando serve

In alternativa è possibile effettuare il redo, solo delle operazioni relative a transazioni committate. Ciò richiederebbe due letture del log: una per la creazione della lista REDO (contenente le transazioni committate), e una per l'effettiva esecuzione del REDO rispetto a tale lista.

Quando il processo di recovery **termina** il sistema diventa nuovamente disponibile.

## 19. DBMS distribuiti

All'interno di un'architettura distribuita, i dati e la computazione sono distribuiti su molteplici macchine. I vantaggi di questa architettura sono:

- Miglioramento delle performance
- Aumento dell'availability
- Aumento dell'affidabilità

Esistono diversi tipi di architetture distribuite:

- **Client/server:** il server gestisce il database, mentre il client gestisce l'interfaccia utente.
- **Distributed database system:** esistono diversi server DBMS installati sui diversi nodi della rete che agiscono autonomamente. I DBMS sono in grado di cooperare ma le proprietà ACID sono più difficili da garantire.
- **Data replication:** una replica dei dati viene memorizzata all'interno di un nodo di rete differente, il server di replica gestisce autonomamente l'update della replica.
- **Architetture parallele:** hanno l'obiettivo di migliorare le performance e possono essere di due tipi
  - Macchine multiprocessore
  - Cluster di CPU

- **Data Warehouse:** server specializzati per il supporto decisionale che eseguono operazioni OLAP

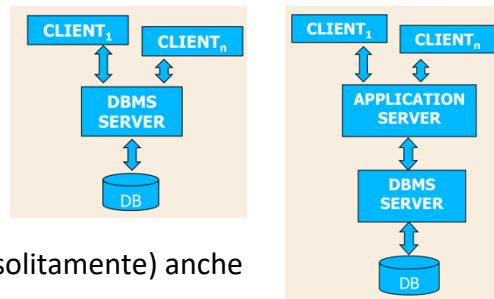
I DBMS distribuiti hanno diverse proprietà:

- **Portabilità:** è la capacità di spostare un programma da un sistema ad un altro, questa proprietà viene garantita dall'SQL standard.
- **Interoperabilità:** è la capacità dei diversi server DBMS di cooperare su specifici task sfruttando protocolli come OBDC e X-Open-DTP.

## Architetture client/server

Possono essere di due tipi:

- **2-Tier:** è composto da:
  - **DBMS server:** fornisce l'accesso ai dati
  - **Thick client:** contiene anche della logica applicativa
- **3-Tier:** è composto da:
  - **DBMS server:** fornisce l'accesso ai dati
  - **Server applicativo:** implementa la logica applicativa e (solitamente) anche un web server
  - **Thin client:** browser



Si possono distinguere due modalità di esecuzione delle query SQL:

- **Compile & Go:** dopo che la query viene ricevuta, il server genera il piano di esecuzione, esegue **direttamente** la query e restituisce il risultato.  
È efficace per l'esecuzione one-shot delle query.
- **Compile & Store:** dopo che la query viene ricevuta, il server genera il piano di esecuzione e lo **memorizza** per possibili utilizzi futuri, in seguito esegue la query e restituisce il risultato.  
È efficace per l'esecuzione ripetuta di query parametriche.

## Sistemi di database distribuiti

Questo tipo di sistemi hanno diverse caratteristiche:

- Una transazione può dover accedere a **diversi** server DBMS
- **Local autonomy:** ogni nodo gestisce i suoi nodi in maniera autonoma

Ci sono diversi vantaggi:

- **Localizzazione** appropriata dei dati e delle applicazioni (anche in base alla distribuzione geografica dei dati)
- Aumento dell'**availability** dovuto al fatto che potrebbero essere più richiesti blocchi locali memorizzati all'interno del DBMS interrogato
- Miglioramento della **scalabilità** causato dalla modularità e dalla flessibilità di questo tipo di architettura

## Design di database distribuiti

Questi sistemi sfruttano la **frammentazione** dei dati, data una relazione R, un frammento può essere di diversi tipi:

- **Orizzontale**: contiene un subset di tuple che hanno gli stessi attributi di R, è ottenuta mediante una **selezione**  $\sigma_p$  (con p predicato di partizionamento), questi frammenti **non sono sovrapposti**. La ricostruzione di R avviene mediante l'**unione** di tutti i frammenti
- **Verticale**: contiene un subset di colonne, è ottenuta mediante una **proiezione**  $\pi_X$  (con X un subset di attributi di R) a cui viene aggiunta la chiave primaria che consente la ricostruzione di R. In questo caso vengono incluse tutte le tuple e i frammenti si sovrappongono sulla chiave primaria. La ricostruzione avviene mediante un'operazione di **join**.
- Ibrida

Due proprietà note della frammentazione sono:

- **Completezza**: ogni unità informativa presente all'interno di R è presente in almeno un frammento  $R_i$ .
- **Correttezza**: il contenuto informativo di R può essere ricostruito a partire dai suoi frammenti.

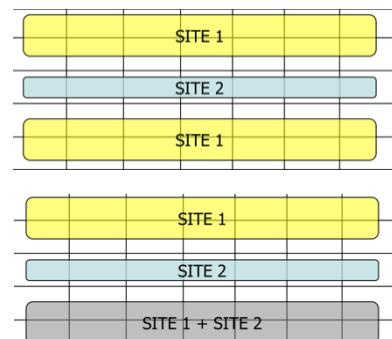
I DBMS distribuiti si basano sull'utilizzo della **frammentazione per distribuire i dati fra i diversi nodi** della rete, in quanto ogni frammento viene salvato in file diversi, memorizzati su server diversi.

Lo **schema di allocazione** descrive come i frammenti sono distribuiti nei diversi nodi:

- **Mapping non ridondante**: ogni frammento è memorizzato su un singolo nodo
- **Mapping ridondante**: i frammenti sono replicati su server differenti che consente una maggiore data availability, ma necessita di una complessa gestione per mantenere le repliche aggiornate.

I **livelli di trasparenza** descrivono il livello di conoscenza relativo alla distribuzione dei dati:

- **Fragmentation transparency**: le applicazioni conoscono l'esistenza delle tabelle ma non ne conoscono i frammenti
- **Allocation transparency**: le applicazioni conoscono l'esistenza dei frammenti, ma non dove essi sono allocati e se essi sono replicati
- **Language transparency**: le applicazioni conoscono sia i frammenti che la loro allocazione, il programmatore deve specificare sia il frammento, che la sua allocazione.



```
SELECT SName
FROM S
WHERE S#=:CODE
SELECT SName
FROM S1
WHERE S# = :CODE
IF(NOT FOUND)
 SELECT SName
 FROM S2
 WHERE S# = :CODE
SELECT SName
FROM S1@xxx.torino.it
WHERE S# = :CODE
IF(NOT FOUND)
 SELECT SName
 FROM S2@xxx.roma1.it
 WHERE S# = :CODE
```

Una **transazione** può essere classificata in diversi modi:

- **Richiesta remota**: richiesta di lettura ad un singolo server remoto
- **Transazione remota**: esecuzione di qualsiasi operazione su un singolo server remoto

- **Transazione distribuita:** esecuzione di qualsiasi operazione, in cui ogni operazione viene fatta su un singolo server. Necessita di essere gestita rispettando l'atomicità globale del db (protocollo 2 phase commit).
- **Richiesta distribuita:** ogni operazione può essere riferita a dati memorizzati su server diversi, solo in questa classe è utilizzata la fragmentation transparency.

## Tecnologie per i DBMS distribuiti

Proprietà **ACID**:

- **Atomicità:** richiede tecniche distribuite come il 2 phase commit
- **Consistenza:** i vincoli vengono imposti solo localmente
- **Isolamento:** richiede 2PL e 2 phase commit
- **Durability:** richiede l'estensione delle procedure locali per gestire l'atomicità in presenza di failure

L'ottimizzazione delle query distribuite viene eseguita dal DBMS che riceve la query, esso:

- Partiziona la query in subquery, ognuna delle quali indirizzate ad un unico DBMS
- Sceglie il piano di esecuzione
- Coordina le operazioni e lo scambio di informazioni fra i nodi interessati

La gestione dell'**atomicità** si basa sul fatto che tutti i nodi che partecipano ad una transazione distribuita devono prendere le stesse decisioni che vengono coordinate dal protocollo **2 phase commit**.

Il protocollo **2 phase commit** consente di coordinare la commit di una transazione distribuita che ha un **Transaction Manager (TM)** e diversi **Resource Managers (RM)** che prendono parte alla transazione e hanno un log personale.

I record all'interno di un **TM** possono essere di:

- **Prepare:** contiene la lista (Node ID + Process ID) dei **RM** che partecipano alla transazione, indica che la transazione è pronta a fare **commit**.
- **Global commit/abort:** decisione finale sull'esito della transazione.
- **Complete:** indica la fine del protocollo

I record all'interno di un **RM** possono essere di:

- **Ready:** l'RM è pronto ad eseguire un'operazione di commit, questa "decisione" non può cambiare e da questo momento in poi l'RM perde l'autonomia sulla transazione corrente. Questo stato può essere salvato nel log e inoltrato al TM solo se il nodo si trova in uno stato affidabile (WAL, commit precedence e le risorse devono essere bloccate).
- **Commit/Abort**

Il protocollo **2 phase commit** è diviso in due fasi:

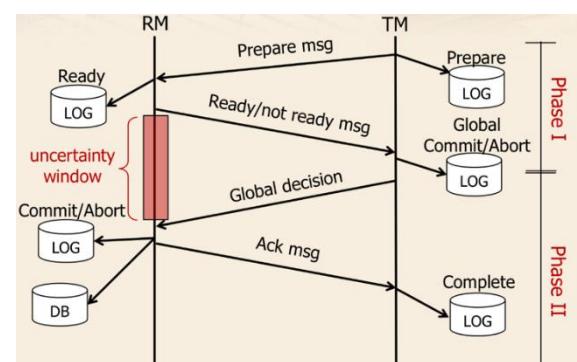
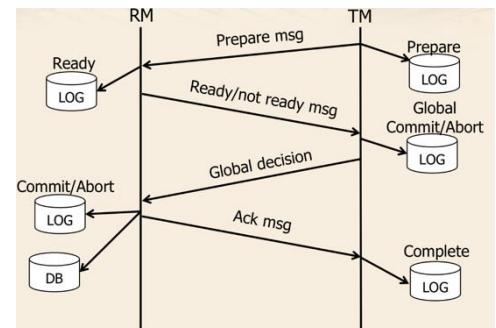
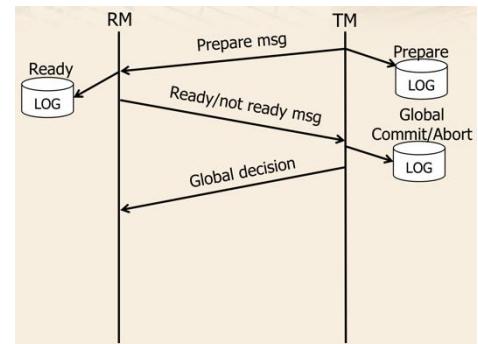
- Fase 1:
  - Il **TM** scrive il **prepare** record sul suo log, invia il **prepare** a tutti gli **RM** partecipanti e setta un timeout nel quale rimane in attesa di una risposta.
  - Gli **RM** dopo aver ricevuto il **prepare**:
    - **Stato affidabile**: scrivono il record di **ready** nel log e inviano il **ready** al TM
    - **Stato non affidabile**: inviano un messaggio di **not ready** al TM, terminano il protocollo ed eseguono un rollback locale
    - **RM spento**: non viene inviata alcuna risposta
  - Il **TM** memorizza tutti i messaggi in ingresso:
    - Se ha ricevuto **ready** da tutti gli RM, scrive sul log il record di **global commit**
    - Se riceve almeno un messaggio di **not ready** o il **timeout** termina, scrive il record di **global abort**
- Fase 2:
  - Il **TM** invia la decisione globale a tutti gli **RM**, e setta un timeout per la ricezione delle risposte
  - Gli **RM** dopo aver ricevuto la decisione globale scrivono ed eseguono il **commit/abort** (ricevuto) e inviano un messaggio di **ack** al TM.
  - Il **TM** memorizza tutti i messaggi in ingresso:
    - Se riceve **tutti** gli **ack** allora scrive il record di **complete** sul log
    - Se mancano alcuni **ack** allo scadere del timeout, la decisione globale viene reinviata e viene resettato il timeout. Questa operazione viene rieseguita fino a che non vengono ricevuti tutti gli **ack**.

Ogni RM ha una **finestra di incertezza** che parte dal momento in cui viene mandato il **ready** e finisce alla ricezione della decisione globale.

Durante questo tempo le **risorse** locali del RM sono **blocate**; quindi, deve essere il più piccolo possibile.

Failure:

- In caso di **failure di un RM** la procedura di warm restart viene modificata aggiungendo un nuovo caso: se l'ultimo record nel log, relativo alla transazione T, è "ready" allora non è nota la decisione globale relativa a T.
- In fase di **recovery** viene creata la **READY** list contenente l'id di tutte le transazioni in ready. Al **restart**, per ognuna delle transazioni nella lista, viene **richiesta la decisione globale** al relativo TM (**remote recovery request**).
- In caso di **failure del TM** in fase di recovery:
  - Se l'ultimo record nel log del TM è **prepare**, viene inviato il **global abort**.
  - Se l'ultimo record è la **decisione globale**, viene ripetuta la fase 2.
- In caso di failure della rete:



- Se avviene in **fase 1**: i messaggi di **prepare** o **ready** non vengono ricevuti, viene eseguito un **global abort**
- Se avviene in **fase 2**: la **decisione globale** o gli **ack** non vengono ricevuti, viene ripetuta la fase 2 dall'inizio.

## X-Open-DTP

X-Open-DTP è un protocollo per la coordinazione di transazioni distribuite basato su **un client, un TM e diversi RM**, che garantisce interoperabilità delle transazioni distribuite su DBMS **eterogenei** (software DBMS diversi).

## Beyond relational databases (NoSQL)

Le caratteristiche principali di un database **NoSQL** sono:

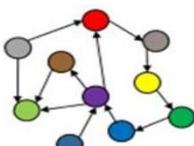
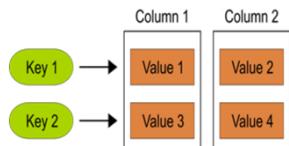
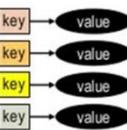
- L'assenza di join
- L'assenza di schema
- La scalabilità orizzontale
- Utilizza un linguaggio custom, ottimizzato per la tipologia di NoSQL utilizzato
- È adatto per dati complessi

Esistono 4 tipi di database NoSQL:

- **Key-value**: è la tipologia più semplice e veloce di database NoSQL, mettendo una chiave ad un valore, alcuni esempi sono: Redis, Riak, Memcached
- **Column-oriented**: memorizza i dati in formato **colonnare**, in cui ogni colonna rappresenta un attributo.  
Le coppie chiave valore vengono memorizzate e recuperate mediante una chiave su sistemi paralleli (come gli indici) la cui divisione risulta invisibile all'applicazione.  
Le righe possono essere ricostruite recuperando i singoli valori delle colonne.  
Alcuni esempi sono: Cassandra, Hbase, Hypertable, Amazon DynamoDB
- **Graph**: si basa sulla teoria dei **grafi** costituiti da **nodi** e **collegamenti**. Questo tipo di database viene utilizzato quando è importante analizzare le relazioni tra i diversi oggetti (amicizie su Facebook).  
Alcuni esempi sono: Neo4J, Infinite Graph, OrientDB.
- **Document**: il database è costituito da documenti che rappresentano una lista di attributi chiave-valore, in cui i valori possono essere costituiti da altri documenti. Per questo motivo la struttura è **gerarchica e auto-descrittiva**.  
Alcuni esempi sono: MongoDB, CouchDB, RavenDB.

**Couch DB** (Cluster Of Unreliable Commodity Hardware) è un esempio di database NoSQL che ha diverse caratteristiche:

- Database **document-oriented** interrogabile e indicizzabile mediante **MapReduce**
- Offre **replicazione** incrementale con identificazione e risoluzione bidirezionali dei conflitti fra i diversi nodi.



- È scritto in **Erlang**, un linguaggio di programmazione funzionale ideale per la produzione di **sistemi distribuiti e concorrenti** che permette **scalabilità**.
- Offre delle API JSON RESTful che ne permettono l'**accessibilità** da qualsiasi ambiente che supporti richieste HTTP.

**MapReduce** è un modello di programmazione distribuito e scalabile per l'elaborazione di big data e consiste nello spostare la computazione ai nodi distribuiti, per alleggerire il carico del nodo che sta eseguendo la query.

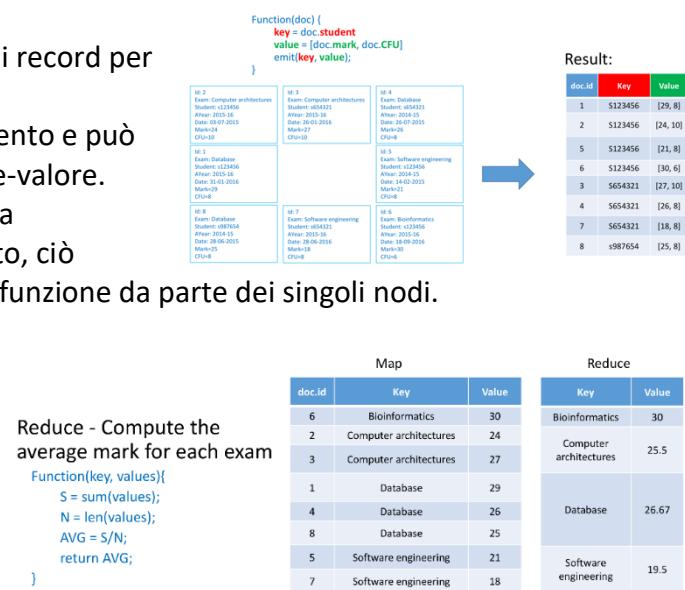
Consiste in due funzioni:

- Map (select+where):** vengono processati i singoli record per avere coppie chiave-valore.

Questa funzione viene chiamata per ogni documento e può saltarlo o estrarre solo alcune delle coppie chiave-valore.

La funzione di Map non dipende in alcun modo da informazioni presenti esternamente al documento, ciò consente l'indipendenza di esecuzione di questa funzione da parte dei singoli nodi.

- Reduce (group by):** a partire dalle coppie chiave-valore ricevute dalla funzione map, a seguito di un'operazione di aggregazione, viene generata una coppia chiave-valore per ogni chiave distinta.



Mediante **re-reduce** è possibile richiamare ricorsivamente la funzione reduce su una chiave composta da un numero di attributi via via minore.

| DB     |                                                                                                             | Map    |                                                                                                            | Reduce |                                   | Rereduce |                        |       |
|--------|-------------------------------------------------------------------------------------------------------------|--------|------------------------------------------------------------------------------------------------------------|--------|-----------------------------------|----------|------------------------|-------|
| doc.id |                                                                                                             | doc.id | Key                                                                                                        | Value  | Key                               | Value    | Key                    | Value |
| 6      | Exam: Database<br>Student: s123456<br>AYear: 2015-16<br>Date: 31-01-2016<br>Mark:29<br>CFU:8                | 8      | Exam: Database<br>Student: s087654<br>AYear: 2016-15<br>Date: 28-06-2015<br>Mark:25<br>CFU:8               | 30     | [Bioinformatics, 2015-16]         | 30       | Bioinformatics         | 30    |
| 2      | Exam: Bioinformatics<br>Student: s123456<br>AYear: 2015-16<br>Date: 18-09-2016<br>Mark:30<br>CFU:8          | 4      | Exam: Computer architectures<br>Student: s087654<br>AYear: 2015-16<br>Date: 26-07-2015<br>Mark:26<br>CFU:8 | 24     | [Computer architectures, 2015-16] | 25.5     | Computer architectures | 25.5  |
| 3      | Exam: Computer architectures<br>Student: s123456<br>AYear: 2015-16<br>Date: 31-01-2016<br>Mark:27<br>CFU:10 | 7      | Exam: Software engineering<br>Student: s123456<br>AYear: 2014-15<br>Date: 28-06-2016<br>Mark:18<br>CFU:8   | 27     | [Database, 2014-15]               | 25.5     | Database               | 27.25 |
| 1      | Exam: Computer architectures<br>Student: s123456<br>AYear: 2015-16<br>Date: 31-01-2016<br>Mark:29<br>CFU:8  | 5      | Exam: Software engineering<br>Student: s087654<br>AYear: 2014-15<br>Date: 28-06-2016<br>Mark:26<br>CFU:8   | 29     | [Database, 2015-16]               | 29       | Software engineering   | 19.5  |
| 8      | Exam: Software engineering<br>Student: s123456<br>AYear: 2014-15<br>Date: 14-02-2015<br>Mark:21<br>CFU:8    | 8      | Exam: Software engineering<br>Student: s123456<br>AYear: 2014-15<br>Date: 28-06-2016<br>Mark:18<br>CFU:8   | 25     | [Software engineering, 2014-15]   | 25.5     |                        |       |
| 5      | Exam: Software engineering<br>Student: s087654<br>AYear: 2014-15<br>Date: 14-02-2015<br>Mark:21<br>CFU:8    | 7      | Exam: Software engineering<br>Student: s123456<br>AYear: 2015-16<br>Date: 28-06-2015<br>Mark:18<br>CFU:8   | 21     | [Software engineering, 2015-16]   | 18       |                        |       |

Seguendo questo approccio è possibile modificare le funzioni di **map** e **reduce** in maniera indipendente.

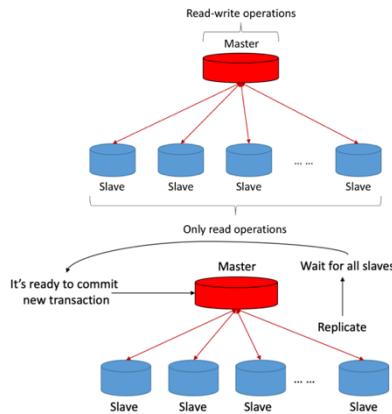
In CouchDB l'unica possibilità per scrivere una query è mediante le **view** (prodotte da MapReduce) che contengono l'id del documento come **key** e l'intero documento come **value**. L'obiettivo è di costruire un indice che memorizza dati correlati mediante chiavi vicine.

La **replicazione** consiste nel memorizzare gli **stessi** dati in posti **diversi**, salvando quindi una porzione dell'intero dataset (chunk) in nodi diversi.

Ciò con l'obiettivo di mantenere i dati in caso di failure di uno dei nodi e di avere, in generale, delle performance migliori.

L'approccio per la replicazione può essere di tipo:

- **Master-Slave:** in questo approccio il server **master** esegue operazioni di write, update e insert mentre uno o più server **slave** attuano le letture. Questo metodo porta ad una **scalabilità solo in lettura** mentre il master rappresenta l'**unico point of failure**. (CouchDB supporta la replicazione Master-Master)
  - **Sincrono:** prima di eseguire un'operazione di commit, il master attende che tutti gli slave facciano commit, per assicurarsi che in caso di failure, tutte i dati siano già stati scritti sugli slave. Questo approccio è poco ottimizzato.
  - **Asincrono:** il master esegue il commit localmente non attendendo gli slave. Ogni slave, in maniera indipendente, applica le modifiche provenienti dal master. Se il master ha una **failure** durante la fase di aggiornamento degli slave alcuni dati possono essere persi. Questo approccio è più **veloce** ma meno **affidabile**.



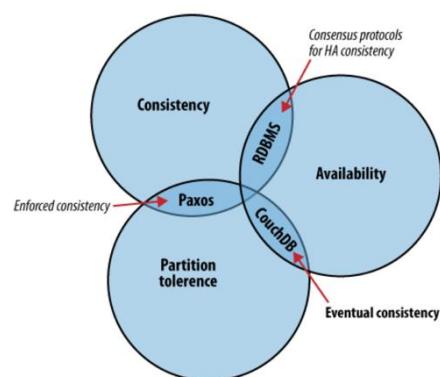
Un **database distribuito** consiste in un insieme di macchine autonome, che lavorano insieme per gestire un dataset **comune**.

Le tre **criticità** tipiche dei database distribuiti sono:

- **Consistency:** tutti i database distribuiti forniscono gli stessi dati all'applicazione
  - **Availability:** una failure non blocca il normale funzionamento degli altri nodi (il servizio rimane disponibile per i clienti)
  - **Partition tolerance:** il sistema continua ad operare anche in caso di perdita di messaggi dovuta a problemi di connessione che vanno a partizionare la rete

Il **teorema CAP** anche detto teorema di **Brewer** afferma che è impossibile assicurare la risoluzione di tutte e tre le criticità precedenti in maniera simultanea in un sistema distribuito.

Per spiegare il CAP prendiamo due nodi nella rete e consideriamo che essi siano scollegati (P) se diamo la possibilità ad un nodo di fare operazioni di aggiornamento (A) sul database i nodi diventeranno inconsistenti (no C), se volessimo invece preservare la consistenza (C), uno dei due nodi deve diventare non disponibile (no A) non permettendogli di eseguire operazioni di update che renderebbero il database inconsistente. Solo quando colleghiamo i nodi (no P) è possibile preservare sia la consistenza (C) che l'a-



In generale per sistemi estesi, i designer non possono non considerare P e devono scegliere quindi su cosa sacrificare fra C e A.

- **Consistenza locale** (CA senza P) (caso centralizzato): il sistema non gestisce il caso in cui si verifichi un partizionamento, in questo caso è sempre possibile assicurare la consistenza e l'availability

- **Transaction locking** (CP senza A) (caso relazionale): in questa situazione il sistema può tollerare inconsistenze e partizionamento semplicemente perché vengono bloccate tutte le richieste venendo meno l'availability. Quest'ultima potrà essere ripristinata alla risoluzione del malfunzionamento.
- **Best effort / Optimistic locking** (AP senza C) (caso non relazionale): in questa situazione non siamo interessati ad una consistenza globale in ogni istante, per questo motivo ogni parte del sistema rende disponibile ciò di cui è a conoscenza. Quando verrà eliminato il partizionamento, allora il sistema può tornare globalmente consistente.

Non è possibile avere C, A o P al 100% (tutte le connessioni vanno giù), per questo motivo anche la visione “2 di 3” non è completamente corretta, per vari motivi:

- Dato che il partizionamento è raro, è poco ragionevole rinunciare a C o A quando il sistema non è partizionato
- La scelta fra C e A può essere presa diverse volte all'interno dello stesso sistema da parti del sistema differenti, o dalla stessa zona del sistema in momenti differenti in quanto la scelta dipende dall'operazione che si sta eseguendo.
- C, A e P non sono “scelte” binarie ma variano in maniera continua da 0% a 100%, in quanto ci possono essere diversi livelli di consistenza e il partizionamento può avere diverse “sfumature” (incluso anche il fatto che all'interno del sistema non tutti sono a conoscenza dell'esistenza della partizione).

Nel mondo non relazionale è più comune considerare le proprietà **BASE** piuttosto che quelle **ACID**:

- **Basically available**: il sistema fornisce availability, dal punto di vista del teorema CAP
- **Soft state**: indica il fatto che lo stato del sistema può cambiare nel tempo, anche senza alcun input, a causa dell’“eventual consistency”.
- **Eventual consistency**: indica che il sistema inizia a diventare consistente, quando smette di ricevere novi input.

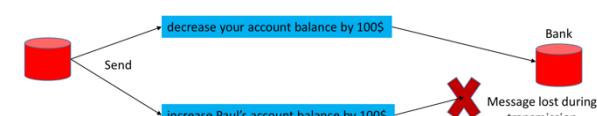
Le proprietà BASE sono concentrate sul massimizzare l'availability, tralasciando la consistenza, un esempio è DNS.

Delle modifiche in conflitto, fatte da due nodi diversi portano ad inconsistenza. Per affrontare questo problema CouchDB utilizza un numero di versione (revision number) su ogni documento e sceglie mediante un algoritmo **deterministico** qual è il “vincitore”. Nello specifico CouchDB crea un hash dei due documenti, e prende in considerazione il primo in ordine ASCII.

In questo modo, quando i due nodi diventeranno consistenti la scelta resterà in qualunque caso la stessa (deterministico).

**Hadoop** è una piattaforma commercializzata da Apache che consente, mediante un file system distribuito, la gestione di grandi moli di dati distribuiti in maniera decentralizzata su diversi cluster. Utilizza yarn per la gestione delle risorse e sfrutta MapReduce (anche a partire da una query SQL, mediante plugin) per la lettura dei dati.

**(Design recipe)** All'interno dei database NoSQL risulta conveniente rappresentare qualsiasi tipo di transazione tramite un documento contenente mittente, destinatario e quantità. Se



invece andassimo a gestire la transazione come due operazioni separate ciò potrebbe portare a inconsistenze.

## MongoDB

**MongoDB** è un database NoSQL document-based open source, flessibile e performante. I termini utilizzati nel mondo SQL vengono traslati come:

- Table → **Collection**
- Record → **Document**
- Column → **Field**

I record vengono rappresentati tramite documenti BSON (una rappresentazione dei documenti JSON in binario).

```
{
 _id: ObjectId("5099803df3f4948bd2f98391"),
 name: { first: "Alan", last: "Turing" },
 birth: new Date('Jun 23, 1912'),
 death: new Date('Jun 07, 1954'),
 contribs: ["Turing machine", "Turing test", "Turingery"],
 views : NumberLong(1250000)
}
```

Ogni documento presenta un campo **\_id** che rappresenta la chiave primaria autogenerata.

Ogni istanza di MongoDB può contenere **diversi database** composti da diverse **collezioni** e ogni collezione contiene una serie di documenti (che possono avere una **struttura** e dei **field diversi**).

Relativamente alla gestione dei database e delle collezioni esistono diversi comandi:

- *show databases*: per mostrare tutti i database presenti
- *use <database-name>*: per utilizzare il db selezionato e crearlo se non esiste
- *db.dropDatabase*: per eliminare il database corrente
- *db.createCollection(<collection-name>,<options>)*: per creare una collezione
- *show collections*: per mostrare tutte le collection presenti nel database
- *db.<collection-name>.drop()*: per eliminare una collezione

Operazioni crud:

- Create:
    - *db.<collection-name>.insertOne(<documento formato da coppie chiave-valore>)*: i valori delle coppie possono essere una array che documenti
    - *db.<collection-name>.insertMany([<lista di documenti separati da virgola>])*
  - Delete
    - *db.<collection-name>.deleteMany(<documento contenente le condizioni da rispettare>)*
- Esempio: *db.people.deleteMany({status:"D"})* elimina le persone con status uguale a "D"
- Read (Query)
    - *db.<collection-name>.find({<conditions>},{<fields-of-interest>})*
      - **Conditions**: un documento che contiene i field e i corrispondenti valori che vogliamo vengano matchati
      - **Fields of interest**: una lista di coppie chiave-valore che hanno come chiave il nome del field e come valore:
        - *1/true*: per includere il field
        - *0/false*: per escludere il field

Esempio: *db.people.find({status: "A"},{user\_id: 1, status: 1, \_id:0})*

In questo esempio vengono mostrati l'*user\_id* e lo status delle persone che hanno status "A".

- `db.<collection-name>.findOne({<conditions>},{<fields-of-interest>})`: uguale al find ma viene restituito solo il primo documento trovato

Per eseguire le operazioni di **join** ci sono due approcci:

- **Object\_ID**: chiamato “Manual Reference”, consiste nel salvare il `_id` di un documento, in un field di un altro documento. Quando quest’ultimo viene richiesto, MongoDB effettua una seconda query che va a recuperare il documento correlato.
- **DBRef**: è uno standard per la rappresentazione di un documento che indica una relazione, contiene tre campi:
  - **\$ref**: contiene l’`_id` della collezione del documento che vogliamo collegare
  - **\$id**: contiene l’`_id` del documento
  - **\$db**: campo opzionale che contiene il nome del database in cui si trova il documento, se non è specificato la collezione si trova nel database corrente

Operatori di comparazione:

- **\$eq** (o `:`): mettta i valori uguali
- **\$gt**: mettta i valori maggiori
- **\$gte**: mettta i valori maggiori e uguali
- **\$in**: mettta i valori che sono nell’array specificato
- **\$lt**: mettta i valori minori
- **\$lte**: mettta i valori minori o uguali
- **\$ne**: mettta i valori diversi da quello specificato, o i documenti che non contengono proprio il campo specificato
- **\$nin**: mettta i valori che non sono presenti nell’array specificato
- **\$exists**: se true mettta solo i documenti che presentano il campo specificato (se false viceversa)
- **\$type**: mettta solo i documenti che hanno il tipo specificato sul campo specificato (il tipo è rappresentato come un numero intero)

Esempio:

`db.people.find({age: { $lt: 25}})`: restituisce le persone con un’età minore di 25

Mentre l’AND viene rappresentato tramite la “,” l’OR viene rappresentato grazie ad **\$or**.

La funzione **find** restituisce un **cursore** che può essere utilizzato per applicare altri filtri e funzione al risultato come ad esempio:

- `cursor.sort()`: ordina il risultato in base all’attributo selezionato specificato come chiave, e 1 o -1 specificati come valore per specificare se l’ordinamento deve essere ascendente o discendente
- `cursor.count()`: restituisce il conteggio dei documenti che hanno fatto match, e può contenere condizioni similariamente a `find`

- **Update**

- `db.<collection-name>.updateOne(<filter>,<update>,<options>)`
- `db.<collection-name>.updateMany(<filter>,<update>,<options>)`

In entrambi i comandi abbiamo:

- **Filter**: condizioni che specificano quali documenti devono essere aggiornati
- **Update**: specifica quali campi devono essere aggiornati e con quali valori

- **Options**

Esempio: vengono modificati size.uom, status e lastModified in tutti i prodotti con una quantità minore di 50.

```
db.inventory.updateMany(
 { "qty": { $lt: 50 } },
 {
 $set: { "size.uom": "in", status: "P" },
 $currentDate: { lastModified: true }
 }
)
```

- **Aggregazioni:** le funzioni di aggregazione possono essere applicate usando il comando `db.<collection-name>.aggregate({<set-of-stages>})`, fra i vari stage sono presenti:
  - **\$match:** filtro utilizzabile anche in uno stadio intermedio, se usato dopo una \$group si comporta come un having
  - **\$group:** gruppà i documenti in input in base al campo indicato e applica le funzioni di aggregazione specificate
  - Tutti i possibili stadi sono elencati [qui](#).

Esempio: dopo aver gruppato per il campo *status* ed aver calcolato la somma, consideriamo solo i gruppi con *total* maggiore di 1000.

```
db.orders.aggregate([
 { $group: { _id: "$status", total: { $sum: "$age" } } },
 { $match: { total: { $gt: 1000 } } }
])
```

- **Indici:** normalmente MongoDB esegue uno scan di ogni documento all'interno della collezione quando esegue operazioni di lettura.

Gli indici sono delle strutture di dati che contengono una piccola porzione dei dati della collezione e ne rendono efficiente la lettura, infatti memorizzano valori ordinati di uno specifico field e il collegamento al documento originale, ciò consente di eseguire in maniera più veloce match di uguaglianza, range e operazioni di ordinamento.

MongoDB crea nativamente un indice su `_id` durante la creazione della collezione.

Per creare un nuovo indice si utilizza il comando:

```
db.<collection-name>.createIndex(<index-keys>, <options>)
```

Le options possono includere:

- **name:** un nome mnemonico per l'indice
- **unique:** booleano che indica se si accetta l'inserimento di documenti con chiavi duplicate
- Altre opzioni come background, dropDups, etc.

È possibile creare indici di diversi tipi:

- **A campo singolo**
- **Composto**
- **Multi-chiave:** per indicizzare il contenuto degli array
- **Geospaziale**
- **Di testo:** per cercare una stringa nel testo dei documenti

- **Hashed**: viene indicizzato l'hash del valore di un field, supporta match di uguaglianza ma non di range

## 20. ElasticSearch

ElasticSearch è un motore di ricerca e analisi distribuito scalabile ed efficiente costruito su **Lucene** che consente diverse tecniche di ricerca ed esplorazione dei dati:

- Ricerca full-text
- Ricerca strutturata:
- Analitiche

Alcuni esempi che utilizzano ElasticSearch sono GitHub, Wikipedia e StackOverflow.

Le ricerche vengono rappresentate da un documento json che può essere formato anche da dati complessi che possono contenere date, dati geografici, testo, array.

I punti di forza di ElasticSearch sono:

- Scaling orizzontale
- **RESTful API**: che consentono l'interfacciamento con qualsiasi linguaggio di programmazione
- **JSON-based**: così da essere sia machine- che human-friendly
- **Replicazione**: ogni modifica viene storicizzata all'interno dei log che vengono memorizzati su diversi nodi per evitare la perdita di dati

La **colonna** di SQL viene chiamata **field**, i field possono essere di diversi tipi e possono contenere valori multipli dello stesso tipo.

La **riga** di SQL viene chiamata **document**. Un document ha un formato delle righe flessibile ed è identificato univocamente dalla coppia:

- **Index**: in cui è memorizzato il documento
- **Id**: identificativo del documento

La **tabella** di SQL viene chiamata **index** e gli index vengono raggruppati in **cluster** che rappresenta il **database** di SQL.

La parola **index** usata come verbo indica l'inserimento di un document in un index.

Inoltre, quelli che in SQL vengono chiamati **index** in ElasticSearch si chiamano **inverted index**, anche in questo caso servono ad accelerare il processo di lettura ma diversamente da SQL sono ricercabili solo i field che sono "indicizzati".

I concetti principali sono:

- **Mapping**: ElasticSearch genera dinamicamente un mapping fra i dati nella query e quelli nei documenti anche se essi sono di due tipi diversi (ad esempio riconosce un date nella query che è di base una stringa)
- **Analysis**: la stringa di query viene processata e standardizza per poter essere utilizzata nella ricerca

- **Query DSL** (Domain Specific Language): è il linguaggio utilizzato in ElasticSearch

La ricerca può essere di tre tipi:

- **Full text**: trova tutti i documenti che matchano con le keywords e li ordina secondo la loro "rilevanza"
- Ricerca **strutturata** su **field** specifici
- Combinazioni dei due

Se si tratta di tipi di dato **tradizionali** (intero, float, date) il valore nel documento deve matchare esattamente il valore indicato nella query, in questo caso non è necessario il concetto di **rilevanza**. Quando invece la ricerca viene fatta su dati testuali (**full text**) abbiamo bisogno di considerare la **rilevanza** dei documenti rispetto alla query.

Per questo tipo di query è necessario anche considerare il senso di una certa parola all'interno della frase e quindi riconoscere:

- Abbreviazioni
- Singolare, plurale, coniugazione del verbo
- Sinonimi
- Ordine delle parole

Solitamente vengono utilizzati dei plugin appositi per riconoscere questi aspetti.

Nell'indexing **full text** ES crea un inverted index, per ogni campo testuale, che contiene la lista di tutte le parole che compaiono in qualsiasi documento della collezione e, per ogni parola, la lista di tutti i documenti in cui è presente.

Prima di generare la lista di parole da inserire all'interno dell'inverted index, ES esegue una fase **analisi** svolta dall'**Analyzer**:

- **Tokenizzazione**: il blocco di testo viene diviso in termini individuali mediante **Tokenizers**
- **Normalizzazione**: i termini vengono normalizzati in una forma standard (portati alla radice, al singolare, al sinonimo più rilevante, ...)

Per fare ciò l'Analyzer ha diverse funzionalità:

- Filtro dei caratteri: pulisce la stringa prima della tokenizzazione (es. convert & to and)
- Tokenizers: dividono la stringa in parole singole considerando la punteggiatura e gli spazi bianchi
- Filtri dei token: operano sui termini singolarmente:
  - Modificando i termini (es. portandoli alla radice o al singolare)
  - Rimuovendo le parole poco rilevanti (es. congiunzioni)
  - Aggiungendo termini (es. sinonimi)

All'interno di ES un **filter** è utilizzato per la ricerca di tipo esatto, mentre le **query** sono solitamente usate per la ricerca di tipo full-text e forniscono anche informazioni sulla rilevanza del match ottenuto.

I filtri, essendo intrinsecamente più efficienti, vengono solitamente utilizzati per ridurre il numero di documenti che devono essere esaminati, in seguito, mediante una query.

Le query in ES vengono espresse in **Query DSL** scritte in formato JSON nel body di una richiesta HTTP.

Esempio (match query): Cercare tutti i documenti dell'indice department che hanno un field "name" contenente il valore "John".

Esempio 2 (query composta da diversi criteri di matching):

**bool**: serve a specificare che la query è composta

**should**: consiste nella condizione OR

**must**: corrisponde con la AND

**must\_not**: specifica la NOT

```
POST departments/_search
{
 "query": {
 "match": { "name": "John" }
 }
}
```

```
POST departments/_search
{
 "query": {
 "bool": {
 "should": [
 {"match": { "name": "John" }},
 {"match": { "name": "Mark" }}
],
 "minimum_should_match": 1,
 "must": [
 {"match": { "title": "developer" }}
],
 "must_not": [
 {"match": { "lastname": "Smith" }}
]
 }
 }
}
```

La match query può essere utilizzata sia per la ricerca di tipo esatto in cui viene ritornato sempre un punteggio di rilevanza **\_score** pari a 1, sia per la ricerca full-text, in quest'ultimo caso prima dell'esecuzione della query essa viene analizzata dall'analyser e le viene assegnato il punteggio di rilevanza **\_score**.

In una **bool** query invece viene combinato lo **\_score** di ogni **must** e **should** che matchano.

È possibile anche specificare diversi indici in cui effettuare la ricerca, separandoli con una virgola.

```
POST rooms,students/_search {...}
```

Mediane ES è possibile anche **modificare i dati**:

- **Insert**: l'inserimento di un nuovo documento viene fatto mediante una POST che contiene il nome dell'indice e l'id (opzionale) nell'URI e il documento nel body.
- **Update**: i documenti in ES sono immutabili, per essere modificato deve essere reindividizzato. Durante l'update quindi il documento viene recuperato, copiato, eliminato e infine viene indicizzato il nuovo documento.  
Per eseguire l'update si utilizza una PUT request in cui vengono specificati il nome dell'indice, l'id del documento nell'URI e i field da modificare, con il nuovo valore, nel body.
- **Delete**: l'operazione di delete viene eseguita mediante una DELETE request in cui vengono specificati il nome dell'indice e l'id del documento nell'URI.  
L'eliminazione dei documenti non è immediata ma viene eseguita in **background**.

```
POST /index_name/<id>
{
 JSON document
}
```

```
PUT index_name/123/_update
{
 "color" : "red",
}
```

```
DELETE index_name/id
```

In ES la **rilevanza** di un risultato **\_score** viene rappresentata come un numero floating point e calcolato per ogni documento che matcha la query. Il risultato della query viene mostrato in ordine discendente rispettivamente allo **\_score** e il punteggio più alto corrisponde al documento con rilevanza maggiore.

Durante il processo di **scoring**:

- Viene calcolata la similarità fra la query e ogni documento, calcolandone lo **\_score**
- Vengono selezionati gli **hits**, che sono i dieci documenti più rilevanti.

- (Opzionalmente) Viene fatto un **rescoring** dei (100) documenti più rilevanti che hanno matchato la query

Per il calcolo della rilevanza:

- Vengono selezionati i documenti che **matchano** la query
- Viene valutata l'importanza di ogni termine rispetto al documento e alla collezione:
  - L'importanza del termine viene valutata mediante **TF/IDF** (Term frequency/Inverse Document Frequency)
    - **Term frequency**: frequenza del termine all'interno del singolo documento (più è alta, più il termine è rilevante).  
Nella formula "frequency" rappresenta il numero di volte in cui il termine t appare nel documento d.
    - **Inverse document frequency**: frequenza del termine nella collezione (più è alta, meno il termine è rilevante).  
Nella formula "numDocs" rappresenta il numero di documenti all'interno dell'indice e "docFreq" il numero di documenti che contengono il termine.
    - **Field-length norm**: normalizzazione rispetto alla lunghezza del field, in quanto più è lungo il field e minore sarebbe la probabilità che le parole in esso siano rilevanti.  
Nella formula "numTerms" rappresenta il numero di termini nel field.
- La funzione di scoring si basa su una combinazione di questi coefficienti che vengono calcolati quando il documento viene indicizzato.
- Il documento e la query vengono rappresentati in forma vettoriale secondo il **Vector Space Model**, in cui la dimensione rappresenta il numero di termini e ogni elemento contiene il peso di ogni termine calcolato mediante TF/IDF.
- Viene valutata la similarità della rappresentazione vettoriale della query e del documento, per fare ciò viene usata la **cosine similarity** che considera l'angolo fra un documento e la query per calcolarne la similarità.

Per la **scalabilità orizzontale** è importante il concetto di **sharding** che consiste nel suddividere un indice in piccole partizioni (shard) che contengono diversi documenti. Quando i dati vengono scritti in una shard devono essere memorizzati su disco in un segmento Lucene (ogni secondo) prima di diventare disponibili alle query.

Un **cluster** è un insieme di macchine nelle quali vengono memorizzate le diverse shard che possono essere anche replicate per motivi di sicurezza.

Lo sharding dà, a questo tipo di sistemi, la possibilità di scalare e di distribuire le operazioni su tutti i nodi presenti nel cluster, migliorando le performance generali grazie alla **parallelizzazione**, e migliorando anche l'**availability**.

ElasticSearch utilizza un controllo della concorrenza **ottimistico** assumendo che i conflitti siano improbabili, per questo motivo non applica alcun tipo di locking. Se i dati vengono modificati fra lettura e scrittura, l'update fallisce.

È probabile che ci siano diverse repliche di una shard all'interno del cluster, è necessario quindi propagare gli aggiornamenti alle shard, per fare ciò ES implementa un meccanismo di **versioning** mediante **\_version** che evita la sovrascrittura da parte di una vecchia versione del documento.

Le API di modifica o cancellazione accettano anche la versione come parametro di input.

